


2011

Effective Task Transfer Through Indirect Encoding

Phillip Verbancsics
University of Central Florida

 Part of the [Computer Sciences Commons](#), and the [Engineering Commons](#)
Find similar works at: <https://stars.library.ucf.edu/etd>
University of Central Florida Libraries <http://library.ucf.edu>

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation

Verbancsics, Phillip, "Effective Task Transfer Through Indirect Encoding" (2011). *Electronic Theses and Dissertations, 2004-2019*. 1722.
<https://stars.library.ucf.edu/etd/1722>

EFFECTIVE TASK TRANSFER THROUGH INDIRECT ENCODING

by

PHILLIP VERBANCSICS

B.S. University of Central Florida, 2006

M.S. University of Central Florida, 2009

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2011

Major Professor:
Kenneth O. Stanley

© 2011 by Phillip Verbancsics

ABSTRACT

An important goal for machine learning is to transfer knowledge between tasks. For example, learning to play RoboCup Keepaway should contribute to learning the full game of RoboCup soccer. Often approaches to task transfer focus on transforming the original representation to fit the new task. Such representational transformations are necessary because the target task often requires new state information that was not included in the original representation. In RoboCup Keepaway, changing from the 3 vs. 2 variant of the task to 4 vs. 3 adds state information for each of the new players. In contrast, this dissertation explores the idea that transfer is most effective if the representation is designed to be the *same* even across different tasks. To this end, (1) the *bird's eye view* (BEV) representation is introduced, which can represent different tasks on the same two-dimensional map. Because the BEV represents state information associated with positions instead of objects, it can be scaled to more objects without manipulation. In this way, both the 3 vs. 2 and 4 vs. 3 Keepaway tasks can be represented on the same BEV, which is (2) demonstrated in this dissertation.

Yet a challenge for such representation is that a raw two-dimensional map is high-dimensional and unstructured. This dissertation demonstrates how this problem is addressed naturally by the Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) approach. HyperNEAT evolves an *indirect encoding*, which compresses the representation by exploiting its geometry. The dissertation then explores further exploiting the power of such encoding, beginning by (3) enhancing the configuration of the BEV with a focus on

modularity. The need for further nonlinearity is then (4) investigated through the addition of hidden nodes. Furthermore, (5) the size of the BEV can be manipulated because it is indirectly encoded. Thus the resolution of the BEV, which is dictated by its size, is increased in precision and culminates in a HyperNEAT extension that is expressed at effectively infinite resolution. Additionally, scaling to higher resolutions through gradually increasing the size of the BEV is explored. Finally, (6) the ambitious problem of scaling from the Keepaway task to the Half-field Offense task is investigated with the BEV. Overall, this dissertation demonstrates that advanced representations in conjunction with indirect encoding can contribute to scaling learning techniques to more challenging tasks, such as the Half-field Offense RoboCup soccer domain.

To my Mom.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Kenneth O. Stanley for years of hard work and patience in guiding me throughout my graduate career, culminating in this dissertation.

Thanks to my committee members for their collective input into this dissertation and, individually, Dr. Ivan Garibay for lessons in evolutionary computation, Dr. Michael Georgiopoulos for teaching the broad subject of artificial neural networks, and Dr. Gita R. Suktanar for introducing me to the RoboCup domain.

I would also like to thank my mother for her support over the years, including many rides back and forth from campus.

Thanks also to the University of Central Florida's Trustees Doctoral Fellowship and the SMART Scholarship Program, funded by OSD-T&E (Office of Secretary Defense-Test and Evaluation), Defense-Wide / PE0601120D8Z National Defense Education Program (NDEP) / BA-1, Basic Research Grant Number N00244-09-1-0081, for funding the entirety of the my Ph.D. research.

Thank you to the University of Central Florida for nine years of education, from freshman year to these last few months of dissertation work. Special thanks to all my professors over the years, including, but not limited to: Drew Lanier, Arup Guha, Mark Llewellyn, Olcay Kursun, Charles Hughes, and Annie Wu. Thanks to members of the Evolutionary Complexity Research Group (Eplex) at University of Central Florida (<http://eplex.cs.ucf.edu>) for suffering through my many practice talks and providing helpful feedback.

Finally, thank you to friends past and present for making my academic career more enjoyable.

Phillip Verbancsics

University of Central Florida

October 2011

TABLE OF CONTENTS

LIST OF FIGURES	xiii
LIST OF TABLES	xlii
CHAPTER 1 INTRODUCTION	1
1.1 Contributions	4
1.2 Outline	5
CHAPTER 2 BACKGROUND	6
2.1 NeuroEvolution	6
2.2 NeuroEvolution of Augmenting Topologies (NEAT)	8
2.3 Generative and Developmental Systems	11
2.4 Compositional Pattern Producing Networks and HyperNEAT	13
2.5 Task Transfer	18
2.6 RoboCup Soccer	24
2.7 Background Summary	27
CHAPTER 3 APPROACH: BIRD’S EYE VIEW	29
3.1 Bird’s Eye View	29

3.2	HyperNEAT: Learning from the BEV	31
CHAPTER 4 ROBOCUP KEEPAWAY BENCHMARK EXPERIMENTS		36
4.1	RoboCup Keepaway Benchmark	37
4.1.1	Approaches Compared	37
4.1.2	Experimental Setup	39
4.1.3	Results	41
4.2	Task Transfer	42
4.2.1	Keepaway 3 vs. 2 to 4 vs. 3	43
4.2.2	Knight Joust to Keepaway	46
4.3	Summary	49
CHAPTER 5 SUBSTRATE CONFIGURATION EXPERIMENTS		50
5.1	Modularity	50
5.1.1	Modularity in Evolutionary Computation	53
5.1.2	HyperNEAT Thresholding and Link Expression Output	56
5.1.3	Experimental Setup	59
5.1.4	Results	64
5.1.5	Discussion	71
5.1.6	Conclusion	77

5.2	Nonlinearity in the BEV	78
5.2.1	Experimental Setup	80
5.2.2	Results	82
5.2.3	Discussion	84
5.3	Summary	85
CHAPTER 6 EXPERIMENTS IN INCREASING RESOLUTION		87
6.1	Transfer of Field Size and Substrate Resolution	88
6.1.1	Experimental Design	88
6.1.2	Results	89
6.1.3	Discussion	92
6.2	Virtual Infinite Resolution by Generation Online	93
6.2.1	Experimental Design	94
6.2.2	Results	97
6.2.3	Discussion	99
6.3	Phased Continuous Substrate Extrapolation	101
6.3.1	Experimental Design	102
6.3.2	Results	104
6.3.3	Discussion	107

6.4	Summary	109
CHAPTER 7 SCALING COMPLEXITY: HALF-FIELD OFFENSE . . .		112
7.1	Half-Field Offense Domain	112
7.2	Experimental Setup	115
7.3	Results	117
7.4	Half-field Offense Discussion	120
7.5	Summary	124
CHAPTER 8 DISCUSSION AND FUTURE WORK		125
8.1	Representations for Transfer Learning	125
8.2	High-dimensional Structures	126
8.3	Implications for Future Work	128
CHAPTER 9 CONCLUSIONS		132
9.1	Contributions	133
9.2	Conclusion	136
APPENDIX A: C# ROBOCUP SIMULATOR		137
APPENDIX B: PARAMETER VALUES		140
LIST OF REFERENCES		144

LIST OF FIGURES

2.1	A CPPN Describes Connectivity. A grid of nodes, called the ANN <i>substrate</i> , is assigned coordinates. (1) Every connection between layers in the substrate is queried by the CPPN to determine its weight; the line connecting layers in the substrate represents a sample such connection. (2) For each such query, the CPPN inputs the coordinates of the two endpoints, which are highlighted on the input and output layers of the substrate. (3) The weight between them is output by the CPPN. Thus CPPNs, whose internal topology and connection weights are evolved by HyperNEAT, can generate regular patterns of connections.	15
2.2	Example CPPN Describing Connections from a Single Node. A CPPN (b) with inputs $(x_1, y_1, x_2, y_2, bias)$ and output <i>weight</i> has a Gaussian node and five connections. The function is symmetric about x_1 and x_2 (due to the Gaussian) and linear with respect to y_2 . For the given input coordinate $(x_1 = 0, y_1 = 0)$, the CPPN produces $\text{Gauss}(-x_2) - y_2$. The weight pattern from node $(0, 0)$ is shown on the contour map (a). Weight values are indicated by color, with lighter colors denoting positive and darker negative. The function embodied by the CPPN thus encodes a geometric pattern of weights in space.	16

2.3	Visualization of Traditional State Variables in 3 vs. 2 Keepaway. The 13 state parameters that represent the state in the 3 vs. 2 Keepaway task are depicted in this figure. The three keepers are represented by the circles and the takers are represented by the triangles. The state parameters include the distances from each player to the center of the field (marked by the circle with the \times), the distances from the keeper with the ball (denoted by the circle with the $+$) to each other player, the distance from each other keeper to the taker nearest them, and the angles along the passing lanes. This dissertation explores alternatives to this traditional representation.	27
3.1	Alternative Representations of a Soccer Field. Several parameters (a) represent the agent's relationship with other agents on a soccer field (taken from a standard RoboCup representation; [CDF03]). Each distance and angle pair represents a specific relationship of the agent to another agent. The BEV (b) represents the same relationships as paths in the geometric space. A square depicts the agent, circles depict its teammates, and triangles its opponents. The overhead perspective also makes it possible to represent any number of agents without changing the representation.	31
3.2	The Importance of True Geometry. A two-dimensional field transformed into a vector of parameters without any geometry forfeits knowledge of the geometry of the domain.	32

3.3	BEV Implemented in the Substrate. Each dimension ranges between $[-1, 1]$ and the input and output planes of the substrate (a) are equivalently constructed to take advantage of geometric regularities between states and actions. Because CPPNs (b) are an indirect encoding, the high dimensionality of the weights does not affect performance. (The CPPN is the search space.)	33
3.4	Changing the BEV. Two kinds of alterations are depicted in this figure. First, the BEV can be altered by increasing the <i>area</i> of the substrate while maintaining the size of each discrete cell by extrapolating new connection weights associated with previously unseen cells. Second, resolution is increased by increasing the number of cells and shrinking the area represented by each discrete cell. The CPPN automatically interpolates connection weights for the new locations. This way, the BEV allows new forms of transfer to differing field sizes or levels of precision.	35

4.1	Visualizing the BEV Input Layer in 3 vs. 2 Keepaway. The BEV input layer is marked with the positions of keepers, takers and paths. The keeper with the ball is the small square, other keepers are circles, and the takers are triangles. Positive values are denoted by lighter shades (for keepers and paths to keepers) and negative values are denoted by darker shades (for takers and paths to takers). The middle shade represents an input of 0.0, the lightest is +1.0, and the darkest is -1.0. The BEV represents the distances and angles in a geometric configuration, allowing geometric relationships to be exploited by HyperNEAT. Paths represent ball position by converging on that keeper.	40
4.2	Transfer Learning From 3 vs. 2 to 4 vs. 3 Keepaway on a 25m×25m Field. As the performance (averaged over five runs) of the champion in the 3 vs. 2 task improves, the untrained performance in the 4 vs. 3 task also consequently improves from 6.6 seconds to 8.1 seconds without <i>ever</i> training for it. The improvement is positively correlated ($r = 0.87$).	44
4.3	Further Training After Transfer From 3 vs. 2 to 4 vs. 3 Keepaway on a 25m×25m Field. Performance of individuals trained on 3 vs. 2 then transferred to 4 vs. 3 and further trained are contrasted with individuals solely trained on 4 vs. 3. All depicted results are performance in the 4 vs. 3 task. Prior training on 3 vs. 2 and transfer to the 4 vs. 3 enhances keeper performance by beginning in a more optimal area of the search space.	45

4.4	Knight Joust World. In Knight Joust, the player (circle) begins on the side marked <i>Start</i> and must reach the side marked <i>End</i> , while evading the opponent (triangle). The player is given the state information of the distance to the opponent, d , the angle between the opponent and the left side, α , and the angle between the opponent and the right side, β . This state information can similarly be drawn on the substrate of the BEV by marking the position of the player, opponent, the path between them, and the paths to the corners. .	47
4.5	Transfer Results from Knight Joust to Keepaway. Untrained and further training performance averaged over 30 runs is shown. The performance of raw champions from Knight Joust on Keepaway exceeds initial random individual by 0.3 seconds. After one generation, this advantage from transfer increases to 0.6 seconds and at 10 generations the advantage is 1.1 seconds. Thus performance on Keepaway, both instantaneous and with further training, benefits from transfer from the Knight Joust domain with significance $p < 0.05$	48

5.1 Seed CPPN Expressing Global Locality. Initial CPPN topologies imparted with geometric principles can seed evolution. For example, a six-dimensional CPPN can be initialized with three hidden nodes connected by positive (black) and negative (gray) weights. These hidden nodes take as input $x_1 - x_2$ (Δx), $y_1 - y_2$ (Δy), and $z_1 - z_2$ (Δz) respectively and have Gaussian activation functions. The nodes are connected to the LEO, which has a bias of -3 . Because the Gaussian function peaks at 0.0, the expression output value is greater than or equal to 0.0 only when Δx , Δy , and Δz are 0.0. Thus the initial population is seeded with CPPNs that constrain connectivity to respect locality. When seeding in this way, the weight output of the CPPN is included as usual with a set of direct connections from all the inputs (not shown). . . 59

5.2	Substrate Configuration for the Retina Problem. The substrate configuration is identical to the setup of Clune et al. [CBM10]. The substrate consists of four layers, with the inputs at $z = 1.0$, the first hidden layer at $z = 0.75$, the second hidden layer at $z = 0.5$, and the outputs at $z = 0.25$. Feed-forward connections are allowed between neighboring layers. Nodes in each z layer are placed at an x, y -coordinate. The y coordinates are indicated by the different circle patterns. These patterns are solid-line circles for $y = 1.0$, filled black circles for $y = 0.0$ and dotted line circles for $y = -1.0$. The left and right sides of the retina are reflected through symmetric coordinates on the negative and positive sides the x -axis. This configuration provides a definite division between left and right modules at $x = 0$	62
5.3	Histogram of Connections by Distance for the Uniform Threshold in Knight Joust. The uniform threshold applies to all connections equally regardless of distance. Results are averaged over 100 runs. The uniform threshold allows a high degree of connectivity regardless of distance, never dropping below 82%. However, there is a small change of 14% between the extremes of the distance groups.	65

5.4	Histogram of Connections by Distance for the Dynamic Distance Threshold in Knight Joust. The distance threshold increases in value as distance between nodes increases, decreasing the number of expressed values for longer distances (averaged over 100 runs). The distance threshold allows a high degree of connectivity for close connections, but the percentage of expressed weights decreases to 1% for the longest distance group.	66
5.5	Histogram of Connections by Distance for the LEO in Knight Joust. Link expression is generated as a function of geometry by the evolved CPPN with LEO (averaged over 100 runs). On average in Knight Joust, the LEO limits connectivity to 61% for connections of all lengths.	67
5.6	Average Percentage of Correct Classifications for Thresholding Methods. The fraction of correct classifications of the 256 patterns is averaged over 100 runs for each of the approaches. Each run lasts 5,000 generations and consists of a population of 500. The uniform and dynamic distance thresholds converge to sub-optimal values. The non-seeded LEO CPPN threshold is still improving at generation 5,000, but is not significantly outperforming the high uniform threshold. However, both the seeded LEOs exceed other approaches' performance with significance $p < 0.001$, demonstrating the importance of locality to the evolution of modularity.	68

5.7	Fraction of Runs that are Perfect by Thresholding Method. The fraction of 100 runs that are successful in correctly classifying all of the 256 possible patterns is shown for each of the thresholding approaches after 5,000 generations. The seeded uniform and dynamic distance thresholds find the solution the least often, with success rates of 0%. The low uniform threshold is able to find a solution in 2% of the runs. Seeding non-LEO CPPNs with locality provides no extra benefit while the non-seeded LEO CPPN threshold matches the high uniform threshold; both find the solution in 12% of the runs. The global locality seed improves the odds of finding the solution to 67%, demonstrating how essential the concept of locality is not only to achieving a high fitness, but to finding the exact solution. The <i>x</i> -axis locality seed improves upon the locality seed, finding the solution in 91% of the runs, confirming the intuition that the solution's best opportunity for modularity in this domain is along the <i>x</i> -axis.	69
5.8	Typical Connectivity Pattern of Uniform Threshold. The uniform threshold fails to generate modularity. Instead, it consistently produces near fully-connected networks. These patterns are similar to those produced by the LEO without a seed. Optimizing weight patterns overrides the creation of modular structures. Red (light) lines are negative weights, black (dark) lines are positive weights, and line thickness indicates weight strength.	70

5.9	Typical Connectivity Pattern of Dynamic Distance Threshold. The dynamic distance threshold generates apparent modularity, but has difficulty finding the correct weight patterns. The problem is that the dynamic distance threshold impacts not only when connections are expressed, but the weight patterns that are created. Because longer distances mean higher thresholds, higher weight outputs become necessary, resulting in a movement towards extreme output values.	71
5.10	Typical Connectivity Pattern of LEO with Locality Seed. Modularity is commonly found when HyperNEAT-LEO is given the concept of locality. Once modularity is found, the regularities needed to solve the task for each module can be discovered in the weight pattern. This separation is possible because HyperNEAT-LEO specifies the pattern of weights and the pattern of connection expression through different outputs of the CPPN. Thus both can be evolved independently, freeing evolution from the burden of searching for a weight pattern that has both the correct weight settings to solve the problem and the necessary modularity.	72

5.11 Average Number of Left-Right Connections in Retina Solutions. The average count of left-right connections across the final champions of the 100 runs is shown for each of the thresholding approaches after 5,000 generations. The unseeded LEO and low uniform threshold constrain left-right connectivity the least, with averages of 34.95 and 22.74 respectively. The high uniform threshold, dynamic distance threshold, and LEO with global locality seed limit the average left-right connectivity similarly, ranging from 11.25 (LEO with Locality Seed) to 13.6 (dynamic distance threshold). Seeding LEO with x -locality generates the best limitation of left-right connections along the x -axis, averaging only 2.9 connections. Biasing LEO towards locality provides the ability to limit connectivity. The addition of the locality seed significantly decreases the left-right connectivity compared to all other approaches. . . . 73

5.12 Fraction of Retina Solutions that Contain Zero Left-Right Connections. The fraction of final champions of the 100 runs that contain zero left-right connections is shown for each of the thresholding approaches after 5,000 generations. The unseeded LEO and low uniform threshold discover solutions with zero left-right connections the least, at rates of 0.07 and 0.03 respectively. The high uniform threshold and dynamic distance threshold find these types of solutions more often, but still at the low rates of 0.22 and 0.18. Seeding LEO with a locality bias (both global and along the x -axis) significantly improves the rate of finding solutions with zero left-right connections. The global locality seed's rate is 0.64 and the x -locality seed's rate is 0.85. Thus separating the connectivity decision into the LEO and biasing it towards locality allows HyperNEAT to generate modular solutions more frequently. 74

5.13 Average Champion Keepaway Performance of HyperNEAT with and without LEO. The performance of Keepaway champions for each of 60 generations averaged over 100 runs is shown for both standard HyperNEAT and HyperNEAT-LEO. The seeded LEO hold time of 16.07 significantly ($p < 0.01$) outperforms standard HyperNEAT's hold time of 15.18 seconds. Additionally, HyperNEAT-LEO learns faster, reaching standard HyperNEAT's asymptotic performance after seven generations, 14 generations before standard HyperNEAT. Thus the LEO extension with the global locality seed improves learning in the RoboCup Keepaway task. 75

5.14 Average Number of Expressed Connections of the HyperNEAT BEV in Keep-away with and without LEO. The average connection count for the population and champions for each of 60 generations over 100 runs. The maximum number of connections is 160,000. Standard HyperNEAT’s champions are always nearly fully connected, never dropping below an average of 159,000 connections. The average population connectivity in standard HyperNEAT decreases to 116,610 connections initially, but then returns to near full connectivity and, similarly to the champions, does not drop below 159,000 connections again. In contrast, the global locality-seeded LEO’s average population and champion connectivity begin low at 1,887 and 7,360 connections, respectively. The average connectivity of the champions increases gradually to 57,080, which is significantly ($p < 0.001$) less than standard HyperNEAT. Additionally, the population average of HyperNEAT-LEO jumps to 67,402 connections by generation 11 and then remains stable, ending at 66,006 connections, which is again significantly ($p < 0.001$) less than standard HyperNEAT. Thus separating the connectivity decision and biasing it towards locality allows HyperNEAT to constrain connectivity, begin with a minimal connectivity, and gradually add connections over time.

5.15 Two-Dimensional Single-Layer Perceptron and the BEV Equivalent. A traditional single-layer perceptron with inputs x and y (a) can only solve linearly separable problems [MP88] in the x, y -plane. The BEV (b) transforms the two inputs of the single layer perceptron into a grid of inputs that form an x, y -plane. Though the BEV remains linear, it can now create a piece-wise linear approximation of a nonlinear function across the x, y -plane. 80

5.16 Impact of Hidden Layers on Performance in RoboCup Keepaway. Average champion BEV performance of 100 runs at each generation is evaluated in the RoboCup Keepaway task without hidden layers, with one hidden layer, and with two hidden layers. The BEV substrate without hidden layers simply has 400 inputs and 400 outputs. A single hidden layer contains 400 hidden nodes and two hidden layers add 800 hidden nodes. The asymptotic performance of the agents with hidden layers exceeds the performance without hidden layers significantly ($p < 0.01$). The BEV without a hidden layer ends with a performance of 16.07 seconds. A single hidden layer improves this performance to 17.35 seconds and a second hidden layer further increases performance to 17.83 seconds, but no significant difference exists between one and two hidden layers. Interestingly, the increase in complexity and size of the BEV substrate does not negatively impact on the rate at which the task is learned. The BEV substrates with hidden layers do not significantly differ in performance from the BEV without hidden layers, until they exceed the performance without the hidden layer. Thus the addition of hidden nodes positively impacts the ultimate performance of the BEV in RoboCup Keepaway, without negatively impacting the task learning speed.	83
---	----

6.1	3 vs. 2 Transfer Performance To Larger Field Sizes. Transfer to larger field sizes is evaluated by testing an individual trained on a 15m×15m field on larger field sizes (20m×20m and 25m×25m). The BEV substrate is scaled to maintain the area represented by each discrete unit. Results from [SS01] show that policies trained by Sarsa and transferred to larger fields <i>decrease</i> in performance. However, the task is <i>easier</i> on larger fields, as shown by fixed policies (Random, Always Hold, and Hand-Tuned) that <i>increase</i> in performance as field size increases. In contrast to Sarsa, the BEV learns to outperform fixed policies and transfers to larger field sizes, <i>significantly improving</i> performance.	90
6.2	Fixed Substrate BEV and the VIRGO equivalent. The original approach to the BEV (a) discretized the field into fixed regions, which are then marked when objects fall within their designated area. The BEV-VIRGO (b) instead generates nodes only at the exact x, y -coordinates for each object and along the relevant paths. The CPPN is then queried for each potential connection between the dynamically generated nodes. Thus the BEV has a virtually infinite resolution (limited by the precision of the data or hardware), by generating the weights online.	94

6.3	Fixed Substrate Keepaway BEV and Nodes that Impact Solution. The Keepaway BEV (a) consists of a grid of nodes that discretize the field into fixed regions. These regions are then marked when objects fall within their designated area. A fraction of the inputs to this fixed BEV substrate (b) have non-zero values. Instead, a majority of the inputs have a value of zero and thus do not impact the output values. Furthermore, a smaller fraction of the output nodes (c) are queried for their values to make a decision, thus rendering the rest of the outputs unnecessary. For example, in 3 vs. 2 Keepaway with a 20×20 BEV, only three out of 400 output nodes are queried on any single time step. Thus activating the full ANN substrate for each time step results in a significant waste of computation.	96
-----	---	----

6.4 Infinite Resolution Performance in RoboCup Keepaway. Average champion BEV performance of 100 runs at each generation is evaluated in the RoboCup Keepaway task at a fixed resolution of 20×20 and the virtual infinite resolution. The effectively infinite resolution BEV-VIRGO performance (17.41 seconds) exceeds the performance of the fixed substrate BEV performance (16.07 seconds) with significance $p < 0.01$. Thus this result confirms the hypothesis that additional precision through increased resolution enhances the performance of the BEV. Interestingly, the BEV-VIRGO with no hidden layer does *not* significantly differ from the performance of the fixed substrate BEV with one and two hidden layers (17.35 and 17.83 seconds, respectively). This lack of difference may imply that the additional resolution improves the BEV's linear approximation of the nonlinear function. 98

6.5 Average Number of Signals Calculated for the Fixed Substrate and VIRGO.

The average number of signals (i.e. weights activated) per time step of simulation for the population and champion of 100 runs at each generation is tracked in the RoboCup Keepaway task for the fixed resolution of 20×20 and the virtual infinite resolution. The fixed resolution begins with minimally connected ANNs because of the LEO with locality seed and then gradually increases the number of connections in the network. In contrast, VIRGO does not vary largely over the generations or between the population average and champion average. The average number of signals calculated per time step for the population increased from 31,016 to 33,318 and the champion average increased from 29,625 to 34,325. Thus the VIRGO begins with a higher computation cost in signals calculate, but ends with a lower cost compared to the fixed substrate. This trend reflects that the dominating factor in signals calculated per time step for VIRGO is in the CPPN and not the ANN substrate.¹⁰⁰

6.6 Total Number of Signals Calculated for the Fixed Substrate and VIRGO.

The average of the sum total signals calculated across all individuals and all generations for 100 runs is shown for the fixed resolution of 20×20 and the virtual infinite resolution in the RoboCup Keepaway domain. The fixed resolution calculates a total of 352,008,926 signals on average for an entire run of 60 generations. In contrast, VIRGO calculates significantly ($p < 0.001$) fewer total number of signals at 193,975,863. Thus the hypothesis that VIRGO provides a computational advantage in signals calculated in the RoboCup Keepaway task over the fixed substrate is confirmed. 101

6.7	Phased Continuous Substrate Extrapolation Performance in RoboCup Keepaway at Each Resolution. The average champion BEV’s performance over 100 runs trained with phased continuous substrate extrapolation is evaluated at each generation in the RoboCup Keepaway task on the fixed resolutions of 5×5 , 10×10 , 20×20 , and VIRGO. The first 15 generations are trained on 5×5 , generations 15 to 30 are trained on 10×10 , generations 30 to 45 are trained on 20×20 , and the final 15 generations are trained on VIRGO. The correlation coefficient of the training resolution to the 5×5 resolution is $r = 0.36$, to 10×10 is $r = 0.62$, to 20×20 is $r = 0.78$, and to VIRGO is $r = 0.82$. Furthermore, each training resolution increase results in a significant ($p < 0.01$) performance increase in the RoboCup Keepaway task. Thus phased continuous substrate extrapolation can effectively learn the Keepaway task by building upon knowledge gained at lower resolutions.	105
-----	--	-----

6.8	Phased Continuous Substrate Extrapolation Performance in RoboCup Keepaway Compared to the Fixed Substrate and VIRGO. The average champion BEV's performance over 100 runs trained with phased continuous substrate extrapolation is evaluated at each generation in the RoboCup Keepaway task and compared to the performance of the fixed substrate and VIRGO. Phased continuous substrate extrapolation's final performance at the virtually infinite resolution is 16.67 seconds, which is less than the 17.41 seconds attained by training solely on the infinite resolution. Furthermore, the performance is not significantly greater than training only with the fixed 20×20 resolution substrate to reach the performance of 16.07 seconds. Finally, the phased continuous substrate extrapolation slows the speed of training, taking more generations to achieve the same fitness as when not varying the substrate resolution. Thus phased continuous substrate extrapolation is not suited to enhancing learning in this task.	106
-----	---	-----

6.9	Phased Continuous Substrate Extrapolation Champion Signal Counts Compared to Fixed Substrate and VIRGO. Champion BEV's signals calculated per time step averaged over 100 runs in the RoboCup Keepaway task are shown. The phased continuous substrate extrapolation signals are counted at the trained resolutions (5×5 , 10×10 , 20×20 , and VIRGO). The phased approach results in near fully-connected networks for the first 30 generations at the lower resolutions. The maximum number of connections in the first 15 generations is 525 and the average number of connections increases up to 524. This pattern is different from the fixed 20×20 substrate that begins with a small number of connections and gradually increases to 57,080 out of a possible 160,000. In fact, once the resolution is increased to 20×20 in the phased approach, the number of connections decreases from 90,739 to 65,541. Thus the coarser resolutions cause the LEO to reduce constraints on connectivity that must be increased once resolution increases.	108
-----	--	-----

6.10	Total Number of Signals Calculated for Phased Continuous Substrate Extrapolation, Fixed Substrate, and VIRGO. The average of the sum total of weights activated across all individuals and all generations for 100 runs is shown for the phased approach, fixed resolution of 20×20 , and the virtual infinite resolution in the RoboCup Keepaway domain. The fixed resolution approach results in an average of 352,008,926 signals calculated for an entire run of 60 generations. In contrast, VIRGO calculates a total of 193,975,863 signals. Similarly, the phased resolutions calculate a total of 187,811,776 signals on average. Thus the phased continuous substrate extrapolation provides a significant computational savings compared to the fixed substrate approach, but is comparable to the computational cost of VIRGO.	109
------	--	-----

7.1	Half-field Offense Domain Field. The four attackers (represented by darker circles) begin on the left side of the half-field. The five defense (represented by lighter circles) begin with two defenders in the square defined by the offense and the remaining defenders are placed within the penalty box on the right side, including the goalie. The penalty box is defined by the largest rectangular boundary on the right side, the goal box by the next inner boundary, and finally the goal mouth by the inner-most boundary. The offense must keep the ball in play, that is, within the area defined by the midfield and the top, bottom, and right sides. In addition, the offense must keep the defenders from possessing the ball. The ultimate goal for the offense is to successfully complete shots that pass over the right-hand side of the boundary of the goal mouth.	114
-----	--	-----

7.2	Visualizing the BEV Input Layer in Half-field Offense. The BEV input layer is marked with the positions of the offense, defense, goal and paths. The offensive player with the ball is the small square, other offense are circles, the takers are triangles, and the diamond is the goal. Positive values are denoted by lighter shades (for offense, the goal, and paths to offense or goal) and negative values are denoted by darker shades (for defense and paths to defense). The middle shade represents an input of 0.0, the lightest is +1.0, and the darkest is -1.0. The BEV represents the distances and angles in a geometric configuration, allowing geometric relationships to be exploited by HyperNEAT. Paths represent ball position by converging on that offensive player.	116
-----	--	-----

7.3	Task Transfer Performance in RoboCup Half-field Offense. Average champion performance of 100 runs at each generation is evaluated in the RoboCup Half-field Offense task, with and without transfer from Keepaway. The performance of initial random policies (0.016) exceeds the initial performance of transferred policies (-0.010). Thus transfer from Keepaway to Half-field Offense has a negative jumpstart performance. In contrast, transfer improves asymptotic performance from 0.062 to 0.081 with significance $p < 0.05$. It should also be noted that transfer's initial negative impact on performance disappears by generation three. Finally, the total reward achieve through transfer (3.112) significantly outperforms total reward without transfer (2.110). These total reward values produce a transfer ratio of $r = 0.475$. Thus transfer from Keepaway to Half-field Offense improves overall performance.	119
-----	---	-----

7.4 Initial Performance of Transferred Keepaway Policies and Random Policies in Half-field Offense. The policies trained in Keepaway and then transferred to Half-field offense episodes result in a goal in 19% of episodes, being blocked by the goalie in 52%, being intercepted the defense in 16%, and the ball traveling out of bounds in 13%, without additional training in the task. In contrast, champions of the initial random populations (without transfer) end episodes with a goal in 22% of episodes, the goalie catching the ball in 56%, the defense intercepting the ball in 12%, and the ball travelling out of bounds 10% of the time. Half-field Offense policies transferred from Keepaway shoot on goal less often than initial random policies. Interestingly, transferred policies have a better capability of keeping the ball in play, indicating knowledge gained from keeping the ball in play in Keepaway bootstraps into Half-field Offense. . . . 121

7.5	Trained Performance of Transferred Keepaway Policies and Trained Policies without Transfer in Half-field Offense. The policies trained in Keepaway, transferred to Half-field offense, and then further trained end with a goal in 27% of episodes, being blocked by the goalie in 51%, being intercepted the defense in 12%, and the ball traveling out of bounds in 10%. In contrast, champions from training without transfer end episodes with a goal 25% of the time, the goalie catching the ball 53%, the defense intercepting the ball 11%, and the ball going out of bounds 11.0% of the time. Thus transfer enhances the ultimate scoring ability of policies in the Half-field Offense domain. In particular, transfer improves the ability to make successful shots, keep the ball away from the goalie, and keep the ball in play.	122
-----	---	-----

A.1	Visualization of the RoboCup Field in the Newly-Implemented Simulator. The visualizer for the new RoboCup simulator is built-in and may be optionally run at program startup. The visualization draws the field, the players (red and blue colors indicate side), and the ball as a gray circle. The visualization can begin, pause, step the simulation, and disable on-screen animation to reduce processing overhead. The simulator displays current simulator time, that is, the number of cycles executed, and the current simulation speed, which is the multiple of simulated cycles executed in one-tenth of a second. The new simulator provides a fast alternative platform for experimenting with learning in RoboCup.	139
-----	---	-----

LIST OF TABLES

4.1	Average Best Performance by Method. The HyperNEAT BEV holds the ball longer than previously reported best results for neuroevolution and temporal difference learning methods. Results are shown for Evolutionary Acquisition of Neural Topologies (EANT) from Metzen et al. [MEK07], NeuroEvolution of Augmenting Topologies (NEAT) from Taylor et al. [TWS06], and State action reward state action (Sarsa) from Stone and Sutton [SS01].	42
6.1	Average Performance of the Best Individuals at Different Resolutions. The regularities learned by the indirect encoding are not dependent on the particular substrate resolution and may be extrapolated to higher resolutions. Increasing the number of connections in the substrate by a factor of 16 (by doubling the size of each dimension) does not degrade performance; in fact, it even improves it significantly in some cases.	91
A.1	Parameter Settings in Experiments	143

CHAPTER 1

INTRODUCTION

Representation is a critical factor in the ability of any algorithm to learn autonomously [Cla89]. Different representations provide different perspectives to the learning algorithm. While one might be appropriate for learning physical control, another might better suit strategic planning. One key learning capability that is influenced by representation is *task transfer*, which means bootstrapping knowledge gained learning one task to facilitate learning another, related task [Car97, TS07a, TSL07]. Transfer avoids wasted computation by recycling learned policies. Bootstrapping learning not only reduces computational waste, but also allows learning to begin in a more optimal area of the search space in the target task. Thus complex tasks may be learned faster through initial training in a simpler version to increase performance [Car97, Tad08]. This transfer capability becomes important as tasks approach human-level complexity.

Presently, leading approaches to task transfer focus on the *process* of transferring knowledge between tasks [TS07a, TSL07, Tad08, TTM08]. A common theme of such approaches is the transformation of learned policies to fit the target tasks. For example, transferring an artificial neural network (ANN) that takes as inputs parameters associated with objects (e.g. location, size, etc.) to a task with more such objects may require transforming the network by adding inputs for parameters associated with each new object [TSL07]. Yet such transformation can disrupt previous learning, thereby requiring the transformed network to

undergo additional training to regain even its former capabilities within the new scenario. In contrast, this dissertation investigates how to *represent* a policy whose representation need not change (i.e. it remains *static*) between tasks, thus facilitating transfer to complex tasks.

The idea that input (i.e. state) representation might remain static during transfer is plausible because the raw inputs to biological organisms, e.g. vision, remain the same even when new tasks are confronted. When a child graduates from playing Keepaway to full-blown soccer, the number of photoreceptors in the eye does not change. An important observation is thus that humans often exploit such alternative state representations. For example, maps for navigation, construction blue prints, and sports play books are not transformed as objects are added. Instead, additional objects are simply drawn onto the existing representation. Accordingly, Kuipers’ Spatial Semantic Hierarchy (SSH) suggests that such *metrical* representation of the geometry of large-scale space is a critical component of human spatial reasoning [Kui00]. Following this reasoning, this dissertation introduces the *bird’s eye view* (BEV) ANN representation that represents different tasks on the same two-dimensional map. Conceptually, the BEV is a metaphor for an internal representation of the state of the world from above. The BEV places objects into the context of the world geometry, allowing geometric relationships to be more easily learned. In this way, if the task is transferred to a version with more objects, the representation remains the same (i.e. static), significantly simplifying task transfer.

The advantage of such representations is that they remain consistent no matter the number of objects. However, the challenge for the BEV is that representing positions at a fine

granularity requires a high-dimensional statespace. An outgrowth of evolutionary computation designed to address such high-dimensional problems is *indirect encoding*, which compresses the representation of the solution by reusing information. The particular indirect encoding in this dissertation, called a *compositional pattern producing network* (CPPN [Sta07]), represents ANN mappings between high-dimensional spaces by exploiting *regularities* in their geometry, which is well-suited to the BEV. An evolutionary algorithm called Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT [GS08, SDG09, GS10a]) that is designed to evolve CPPNs is thus able to learn effectively from the BEV. Because HyperNEAT evolves an indirect encoding that generates regular patterns, the ANN structures it produces surpass the complexity of traditional ANNs.

This dissertation demonstrates the BEV’s effectiveness [VS10b, VS10a] and explores its potential. In particular, the BEV’s high dimensionality increases the need for sound organizational principles in configuring the ANN. Thus several aspects of ANN configuration are investigated, including the role of hidden nodes in the BEV and the problem of encouraging modularity by constraining connectivity [VS11]. In addition, the size of the BEV can be scaled. For example, the number of input nodes in the ANN control the resolution of the BEV, e.g. a field that is 20m×20m may be represented by a 20×20 BEV has a resolution of 1m². Because the BEV is indirectly encoded, this resolution can be manipulated without altering the learned policies. Thus evolution can begin with a low resolution that is then incrementally increased to improve accuracy. Furthermore, the indirect encoding ultimately means that the BEV can be expressed at effectively infinite resolution. This

infinite-resolution BEV is implemented by creating nodes at *only* necessary positions and dynamically querying the CPPN for their connectivity. Finally, these capabilities are combined to scale to the much harder Half-field Offense RoboCup task [KLS07b], which is a major stepping stone toward machine learning for full RoboCup [KLS07b, KAK97, Mac09]. The hope is that such scaling will suggest the potential of machine learning methods to achieve human-level capability.

1.1 Contributions

The research hypothesis of this dissertation is that a transfer learning method that does not change representation between tasks, such as bird’s eye view, can facilitate transfer. This hypothesis is supported by the following key contributions:

1. The BEV representation is introduced, which is a high-dimensional alternative representation enabled by indirect encoding.
2. A comprehensive study investigates learning with the BEV in the popular RoboCup Keepaway benchmark, compares it to traditional learning approaches (Sarsa, NEAT, and EANT) in directly learning the task, and compares it to the transfer learning approach TVITM-PS in transferring between tasks.
3. HyperNEAT with the Link Expression Output extends HyperNEAT to optimize connectivity separately from weight values and to generate modular patterns, thereby

enabling HyperNEAT to better optimize the high-dimensional structures required by the BEV.

4. Nonlinearity in the BEV is investigated through a comparison of the BEV with and without hidden layers in the RoboCup Keepaway benchmark.
5. Methods are introduced to scale the size of the BEV to address larger tasks, culminating in an approach that enables virtually infinite resolution.
6. HyperNEAT-trained Keepaway policies are transferred to the complex RoboCup Half-field Offense domain.

1.2 Outline

The dissertation begins with background on neuroevolution, generative and developmental systems (including the HyperNEAT method), and task transfer . Next, Chapter 3 introduces the BEV representation. The experiments that demonstrate the BEV’s capabilities in the RoboCup Keepaway domain are disclosed in Chapter 4. Configuring the BEV substrate is next explored in Chapter 5. Chapter 6 then investigates expanding the resolution of the BEV, up to an effectively infinite resolution. Chapter 7 then culminates with scaling to the complex task of Half-field Offense. Finally, general discussion and conclusions are in Chapters 8 and 9.

CHAPTER 2

BACKGROUND

This chapter reviews several topics fundamental to understanding the approach in this dissertation. The first two sections focus on the concepts of neuroevolution and a specific neuroevolution method, NeuroEvolution of Augmenting Topologies. The following two sections review generative and developmental systems and a generative system for ANNs. Task transfer and the challenges faced by transfer methods are examined next. Finally, the challenging machine learning domain of RoboCup is described.

2.1 NeuroEvolution

The class of methods that train ANNs through evolutionary algorithms is called *neuroevolution* (NE; [Yao99, FDM08]), which include genetic algorithms [IJC89, IJC91, WDD93, SM02, DM92], evolutionary programming [ASP00], co-evolution [MM96, GM99], and generative systems [SDG09, GWP96]. As opposed to updating weights according to a learning rule, such as in backpropagation, candidate ANNs are evaluated in a task and assigned fitness to allow selection and creating a new generation of ANNs by mutating and recombining their selected genomes. Through such evolution, new and fitter solutions are found [Yao99].

This method of training ANNs is beneficial in reinforcement learning tasks in which the optimal sequence of outputs is unknown. The diverse methods [ASP00, MM97, GM99, GWP96, SF95, SM02, SDG09, Thi96, FDM08] for evolving ANNs differ based on design decisions such as encoding. A major dichotomy among NE encodings is between those that evolve only the weights of the ANN [WSB90, IJC89, FFP90, MM96, GM99], and those that evolve both the weights and topology [PP98, SM02, SDG09, ASP00, GWP96, DM92, BW93].

Among those methods that evolve only weights, the fixed-topology of the ANN is specified a priori. These fixed topologies can be recurrent, feed-forward, fully-connected, or sparsely connected; however the burden of design is on the experimenter. In addition, instead of evolving the complete ANN, some approaches to evolving fixed topologies evolve populations of neurons that are then combined into the complete ANN [GM99, MM97].

Alternatively, neuroevolutionary methods may evolve both the topology and weights of the ANN [Yao99, FDM08]. Instead of simply encoding the weights, in this case the encoding also specifies the topology of neurons and connections. That way, the structure of the ANN can also be evolved simultaneously with the weights, removing the burden of topology design from the human. Evolving topology provides the means to begin with simple solutions and then increase their complexity as needed [SM02]. Such complexification is an important means of beginning search in a tractable space and then gradually increasing the search space dimensionality, which is helpful in domains that require large ANNs.

Methods that evolve topology face several challenges. One such challenge is selecting an encoding to represent the ANN. A popular choice is to directly encode each parameter

of the ANN in the genome [ASP00, Yao99, SM02]. These parameters can include weights, which nodes to include, and activation functions. Each of these parameters must then be individually optimized, impacting the size of the search space. Alternatively, instead of directly specifying the entire ANN structure, the ANN may be generated *indirectly* through rules or a growth process, such as development, featured in approaches that include cellular encoding [GWP96], compositional pattern producing networks [SDG09, Sta07], L-systems [Lin68, HP02, BK99], gene regulatory networks [Tur52]. In this way, the search space remains lower-dimensional, while still generating large ANNs by reusing genetic information. Such indirect encodings are discussed further in Section 2.3.

Whether direct or indirect, topology-evolving methods face the dual challenges of how to recombine parts of different ANN genomes and the *competing conventions problem*, which means that the same solution can be encoded in different ways [Whi95, Rad93]. For variable-topology NE, this problem is also called the *topology matching problem*. A NE approach that addresses these challenges is described in the next section.

2.2 NeuroEvolution of Augmenting Topologies (NEAT)

NEAT [SM02, SM04] is a popular neuroevolution method for policy search. The basic principles of NEAT supply the foundation of the approach in this dissertation. Traditionally, ANNs evolved by NEAT control agents that select actions based on their sensory inputs. It

is proven in a variety of challenging tasks, including particle physics [Aal09, WW07], simulated car racing [CLL09], RoboCup Keepaway [TWS06, WTS09], function approximation [Whi05], real-time agent evolution [SBM05], and other control and decision-making tasks [SM02, SM04]. This section briefly reviews NEAT; a full treatment can be found in Stanley and Miikkulainen [SM02] and Stanley and Miikkulainen [SM04].

NEAT is an evolutionary algorithm that begins with a population of small, simple ANNs that increase their complexity over generations by adding new nodes and connections through mutation. That way, the topology of the network does not need to be known a priori and NEAT finds a suitable level of complexity for the task. NEAT is unlike many previous methods that evolved neural networks, which historically evolved either fixed-topology networks [GM99, SF95] or arbitrary random-topology networks [ASP00, GWP96, Yao99]. Unlike these approaches, NEAT begins evolution with a population of small, simple networks and increases the complexity of the network topology into *diverse species* over generations, leading to increasingly sophisticated behavior. A similar process of gradually adding new genes has been confirmed in natural evolution [Mar99, WHR87] and shown to improve adaptation in a few prior evolutionary [Alt94] and neuroevolutionary [Har93] approaches. However, a key feature that distinguishes NEAT from prior work in growing ANNs is its unique approach to maintaining a healthy diversity of increasingly complex structures simultaneously, as this section reviews.

The NEAT method is based on three key ideas. First, to allow network structures to increase in complexity over generations, a method is needed to keep track of which gene is

which. Otherwise, it is not clear in later generations which individual is compatible with which in a population of diverse structures, or how their genes should be combined to produce offspring. NEAT solves this problem by assigning a unique *historical marking* to every new piece of network structure that appears through a structural mutation. The historical marking is a number assigned to each gene corresponding to its order of appearance over the course of evolution. The numbers are inherited during crossover unchanged, and allow NEAT to perform crossover among diverse topologies without the need for expensive topological analysis.

Second, NEAT divides the population into species so that individuals compete primarily within their own niches instead of with the population at large. Because adding new structure is often initially disadvantageous, this separation means that unique topological innovations are protected and thus have the opportunity to optimize their structure without direct competition from other niches in the population. The historical markings help NEAT determine to which species different individuals belong.

Third, many approaches that evolve network topologies and weights begin evolution with a population of random topologies [GWP96, Yao99]. In contrast, NEAT begins with a uniform population of simple networks with no hidden nodes, differing only in their initial random weights. Because of speciation, novel topologies gradually accumulate over evolution, thereby allowing diverse and complex phenotype topologies to be represented. No limit is placed on the size to which topologies can grow. New structures are introduced incrementally as structural mutations occur, and only those structures survive that are found

to be beneficial through fitness evaluations. In effect, then, NEAT searches for a compact, appropriate topology by incrementally adding complexity to existing structure.

The important concept for the approach in this dissertation is that NEAT complexifies evolved ANNs, which will be extended to an indirect representation in Section 2.4. The next section reviews generative and developmental systems, which compress solution representations and enable high-dimensional structures to be optimized effectively.

2.3 Generative and Developmental Systems

While the human brain includes trillions of connections [KSJ00, ZBL99] with complex patterns of organization, traditional approaches to building and training ANNs produce networks significantly smaller in size [SDG09, GM99, SM02, Yao99]. Natural biological systems are capable of generating large-scale organized structures by reusing genetic information [BK99, HP02, KSJ00, SM03]. This information reuse allows patterns of repeated structures to be produced and allows the search space to be compressed. Collectively, methods that emulate this ability are known as generative and developmental systems (GDS). This section reviews two traditional approaches, grammatical and cell chemistry systems.

Rule-based grammar-rewriting systems, such as Lindenmayer Systems (L-Systems), are high-level abstractions of the developmental process [GWP96, HP02, Lin68]. In such systems, a sequence of rules is applied to transform a set of parameters. These parameters often

are a string of characters that are iteratively rewritten with new characters according to the set of grammar rules. The rules in effect specify an indirect encoding that can create strings much longer than the original string and set of rules. Further, rewriting characters in this way makes it possible to create repetitive and regular patterns. Such grammatical rewriting systems can generate a variety of structures, from three-dimensional morphologies to ANNs [GWP96, HP02]. However, grammatical encodings can be brittle because a small change in any of the rules can lead to significant idiosyncratic changes in the resulting structure, reducing the ability to optimize the solution.

Another common approach in GDS is to lower the abstraction level of the simulated developmental process by emulating natural cell chemistry [Tur52, Bon02, Egg97]. Through a more low-level abstraction of the developmental process, the hope is to capture the essential mechanisms that enable nature to create complex organisms. This lower-level abstraction simulates the chemical processes that take place on the cellular level, such as the diffusion of the chemical gradients and protein interactions. For example, genetic regulatory networks (GRNs) send signals through the production of proteins, allowing the components of the network to interact. Through these local interactions, structures can be grown indirectly and with regularities according to the rules of interaction. However, such low-level abstractions can also potentially decrease the tractability of the problem because the search must discover how to manipulate the chemical gradients and how to construct the higher-level interactions. Additionally, maintaining a high level of detail exacts significant computational cost, limiting complexity and the time available to search [Sta07].

The next section describes the particular generative encoding that underlies the approach in this dissertation, which provides a different level of abstraction from either grammar-based or chemistry-based systems, yet still enables the indirect encoding of large structures.

2.4 Compositional Pattern Producing Networks and HyperNEAT

Neuroevolution algorithms can be combined with generative and developmental systems to generate ANNs indirectly. As this section describes, NEAT has been extended in such a way to evolve an indirect encoding, which means a *compressed* description of the solution network. Such compression makes the policy search practical even if the state space is high-dimensional. The approach in this dissertation to indirect encoding is to compute the network structure as a function of the domain’s geometry. This section describes this extension of NEAT, called Hypercube-based NEAT (HyperNEAT; [GS08, SDG09, GS10a]). The effectiveness of the geometry-based learning in HyperNEAT has been demonstrated in multiple domains, such as checkers [GS08, GS10a], multi-agent predator prey [DS08, DS10], visual discrimination [SDG09], and quadruped locomotion [CBO09]. For a full HyperNEAT description, see Stanley et al. [SDG09] or Gauci and Stanley [GS10a].

The main idea in HyperNEAT is that it is possible to learn geometric relationships in the domain through an indirect encoding that describes how the *connectivity* of the ANN can be *generated* as a function of the domain geometry. Unlike a *direct* representation,

wherein every dimension in the policy space (i.e. each connection in the ANN) is described individually, an indirect representation can describe a pattern of parameters in the policy space without explicitly enumerating every such parameter. That is, information is reused in such an encoding, which is a major focus in the field of generative and developmental systems from which HyperNEAT originates [BK99, HP02, Lin68, Tur52]. Such information reuse is what allows indirect encodings to search a compressed space. That is, HyperNEAT discovers the *regularities* in the domain geometry and learns a policy based on them.

The indirect encoding in HyperNEAT is called a *compositional pattern producing network* (CPPN; [Sta07]), which encodes the *connectivity pattern* of an ANN [GS07, GS08, SDG09, GS10a]. The idea behind CPPNs is that a geometric pattern can be encoded by a *composition of functions* that are chosen to represent several common regularities. For example, because the Gaussian function is symmetric, when it is composed with any other function, the result is a symmetric pattern. The internal structure of a CPPN is a weighted network, similar to an ANN, that denotes which functions are composed and in what order. The appeal of this encoding is that it can represent a pattern of connectivity, with regularities such as symmetry, repetition, and repetition with variation, through a network of simple functions (i.e. the CPPN), which means that, instead of evolving ANNs directly, NEAT can evolve *CPPNs* that generate ANN connectivity patterns (figure 2.1). Furthermore, the indirect encoding represents the connectivity of the ANN regardless of its size, which allows ANNs of arbitrary dimensionality to be represented.

Formally, CPPNs are *functions* of geometry (i.e. locations in space) that output con-

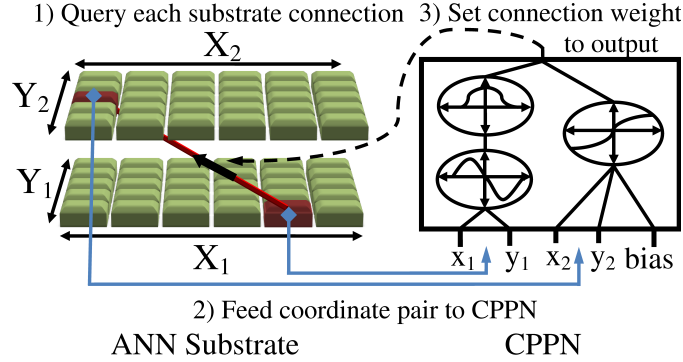


Figure 2.1: A CPPN Describes Connectivity. A grid of nodes, called the ANN *substrate*, is assigned coordinates. (1) Every connection between layers in the substrate is queried by the CPPN to determine its weight; the line connecting layers in the substrate represents a sample such connection. (2) For each such query, the CPPN inputs the coordinates of the two endpoints, which are highlighted on the input and output layers of the substrate. (3) The weight between them is output by the CPPN. Thus CPPNs, whose internal topology and connection weights are evolved by HyperNEAT, can generate regular patterns of connections.

nectivity patterns whose nodes are situated in n dimensions, where n is the number of dimensions in a Cartesian space. For each connection between two nodes in that space, the CPPN inputs their *coordinates* and outputs their connection weight. That way, NEAT can evolve CPPNs that represent ANNs with symmetries and regularities that are computed *directly* from the geometry of the state space. Consider a CPPN that takes four inputs labeled x_1 , y_1 , x_2 , and y_2 ; this point in four-dimensional space can *also* denote the connection between the two-dimensional points (x_1, y_1) and (x_2, y_2) . The output of the CPPN for that input thereby represents the weight of that connection (figure 2.1). By querying every pair of points in the space, the CPPN can produce an ANN, wherein each queried point is the position of a neuron.

While CPPNs are themselves networks, the distinction in terminology between CPPN and ANN is important for explicative purposes because in HyperNEAT, CPPNs *encode*

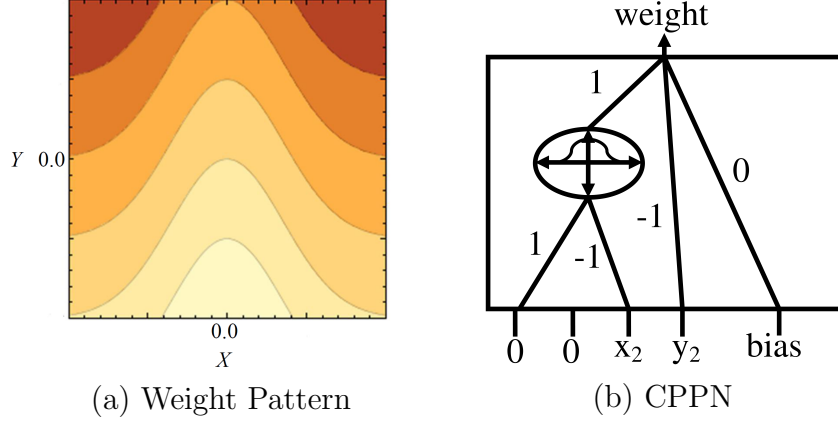


Figure 2.2: Example CPPN Describing Connections from a Single Node. A CPPN (b) with inputs $(x_1, y_1, x_2, y_2, bias)$ and output *weight* has a Gaussian node and five connections. The function is symmetric about x_1 and x_2 (due to the Gaussian) and linear with respect to y_2 . For the given input coordinate $(x_1 = 0, y_1 = 0)$, the CPPN produces $\text{Gauss}(-x_2) - y_2$. The weight pattern from node $(0, 0)$ is shown on the contour map (a). Weight values are indicated by color, with lighter colors denoting positive and darker negative. The function embodied by the CPPN thus encodes a geometric pattern of weights in space.

ANNs. Because the connection weights are produced as a function of their endpoints, the final structure is produced with *knowledge* of the domain geometry, which is literally depicted geometrically within the constellation of nodes. In other words, parameters of the state vector actually exist at *coordinates* in space, giving it a geometry.

To help explain how CPPNs can compactly encode regular connectivity patterns, figure 2.2 shows how a very simple CPPN encodes a symmetric network. In effect, the CPPN paints a pattern within a four-dimensional hypercube that is interpreted as an isomorphic connectivity pattern. The example in figure 2.2 illustrates the natural connection between the function embodied by the CPPN and the geometry of the resultant network. Patterns thus can be elaborated through composition with other functions. For example, combining a sigmoid and Gaussian function generates a symmetric sigmoid function.

Connectivity patterns produced by a CPPN in this way are called *substrates* so that they can be verbally distinguished from the CPPN, whose internal topology is independent of the substrate. The experimenter defines both the location and role (i.e. hidden, input, or output) of each node in the substrate. As a rule of thumb, nodes are placed on the substrate to reflect the geometry of the domain (i.e. the state), which makes the setup straightforward (e.g. positions on a game board [GS08, GS10a], limb-joint locations [CBO09], and sensor-motor locations [SDG09]). This way, the connectivity of the substrate becomes a direct function of the domain geometry, which means that knowledge about the problem can be injected into the search and HyperNEAT can exploit the regularities (e.g. adjacency, or symmetry, which the CPPN sees) of a problem that are invisible to traditional encodings. For example, geometric knowledge can be imparted e.g. by including a hidden node in the CPPN that computes $\text{Gaussian}(x_2 - x_1)$, which imparts the concept of locality on the x -axis, an idea employed in the implementation in this dissertation. The HyperNEAT algorithm is outlined in algorithm 1.

In summary, instead of evolving the ANN directly, HyperNEAT evolves (through the NEAT method) the topology and weights of the CPPN that *encodes* it, which is more compact. Through this compressed solution description, alternative state representations, such as the static representation in this dissertation, may be exploited. The next section explores task transfer, which may benefit from the geometry-based learning of HyperNEAT.

	Input: Substrate Configuration
	Output: Solution CPPN
1	Initialize population of minimal CPPNs with random weights;
2	while <i>Stopping criteria is not met</i> do
3	foreach <i>CPPN in the population</i> do
4	foreach <i>Possible connection in the substrate</i> do
5	Query the CPPN for weight w of connection;
6	if $Abs(w) > Threshold$ then
7	Create connection with a weight scaled proportionally to w (figure 2.1);
8	end
9	end
10	Run the substrate as an ANN in the task domain to ascertain fitness;
11	end
12	Reproduce CPPNs according to the NEAT method to produce the next generation;
13	end
14	Output the Champion CPPN.

Algorithm 1: Basic HyperNEAT Algorithm

2.5 Task Transfer

Task transfer means applying knowledge learned in one task to a new, related task [Car97, TS07a, TSL07]. It allows learning to be recycled instead of starting anew, thereby avoiding wasted computation. Additionally, a task may be so complex that it requires initial training on a simpler version to reduce learning time and increase performance [Car97, SI94]. Thus the capability to transfer is becoming increasingly important as the tasks increase in complexity. However, transfer learning faces several challenges: First, transfer is only effective among compatible tasks and the particular knowledge that can transfer from one task to another must be identified. Second, a method must be derived to actually implement the transfer of knowledge. Finally, cases in which transfer hinders performance, or *negative transfer*, must be avoided [PY08]. There are several types of transfer learning problems and

a variety of methods that exploit their characteristics. These methods include translating the knowledge learned in one task to another task [RDC07, TSL07], choosing the best policy for the current task from a set of previously learned policies [TS07a], extracting advice from previously learned tasks [TSW08, TTM08], and learning multiple tasks at the same time [CW08]. This section reviews metrics for evaluating transfer learning and several such transfer learning approaches.

Different benefits of task transfer can be measured through different metrics. While one metric may determine the immediate benefit of transferred knowledge, another may show the ultimate contribution to final performance. Taylor and Stone [TS09] describe several metrics for task transfer in reinforcement learning domains. These metrics include *jumpstart*, *asymptotic*, *total reward*, *transfer ratio*, and *time to threshold* performance. Each of these measures are suited to particular features of task transfer and do not capture the full complexity of transfer. *Jumpstart* is the performance difference between initial random policies and transferred policies in the target task, measuring the immediate impact of transfer on performance. It does not take into account transfer’s impact on further learning. In contrast, *asymptotic* performance compares the final performance *after* learning the target task. This metric examines transfer’s impact on further learning in the target task by looking at ultimate performance, with and without transfer. The difficulties with such a measure is that it does not take into account the time to reach asymptotic performance, the computational expense in converging to the asymptote (if there even is convergence), or the expectation of a difference in final performances. *Time to threshold* evaluates the acceleration in learning a

target task by measuring the training time difference in reaching a predefined performance threshold, with and without transfer. Measuring *total reward* attempts to aggregate the essence of jumpstart, asymptotic, and time to threshold performances by measuring the total accumulated reward during learning. In this way, starting at a better performance, learning the task faster, and converging to a higher asymptotic performance all contribute to the comparison of task transfer. Finally, *transfer ratio* simplifies comparisons by calculating the ratio, r , between the total reward with transfer, y , and without transfer, x , through the equation $r = (y - x)/x$.

An intuitive approach to transfer learning is to transform the representation of knowledge learned in one task to a suitable form for a new task and then continue learning from that point. A successful method that takes this approach is *transfer via inter-task mapping for policy search methods* (TVITM-PS; [TSL07]). TVITM-PS is such a leading method for transforming the policy learned in the source task into a policy usable in the target task. In TVITM-PS, a transfer functional ρ is defined to transform the policy π for a source task into the policy for a target task, such that $\rho(\pi_{source}) = \pi_{target}$. This functional is often hand-coded based on domain knowledge, though learning it is possible. When there are novel state variables or actions, an *incomplete mapping* is defined from the source to the target. TVITM-PS can be adapted to multiple representations. For example, in an ANN, input or output nodes whose connections are not defined in the mapping (i.e. it is incomplete) are made fully connected to the existing hidden nodes with random weights. This incomplete mapping implies that further training is needed to optimize the policies with respect to the

new state variables and actions. However, it makes it possible to begin in the target domain from a better starting point than from scratch. TVITM-PS is a milestone in task transfer because it introduces a formal approach to moving from one domain to another that defines how ambiguous variables in the target domain should be treated.

Another method of transfer is to recycle the exact *same* policy from a source task in a later target task. The idea is that the policy can then continue to improve in the target task. An existing approach to recycling past policies is to maintain a *set* of policies and select among them. *Alternating trusting Exploration and suspicious exploitation* (AtEase; [TS07a]) is such a transfer method; it aims to recognize when tasks are related and when to exploit knowledge gained from previous tasks. It exploits knowledge from previous tasks by judging when to invoke the previously gained knowledge and from which policies. To facilitate this process, a set of policies previously developed by learning source tasks are first evaluated. This evaluation estimates the performance of these previously learned source policies in the new target task. Second, the strategies are ranked by their expected performance in the target task and the source policy with the best estimated performance is chosen. Finally, the chosen best policy is set as the current policy for the target task. It remains as the policy for the target task until the policy’s actual performance in the task falls below expectation (i.e. the estimated performance from the evaluation of source policies is greater than the current performance) or reaches a maximum number of iterations (allowing other policies to be explored). If the expert policy falls below expectation, the next best policy is selected and is set as the current expert policy. This method allows an accurate estimate of which

policy from previously learned tasks is appropriate for the current task. The difficulty in such methods is that the experimenter must take care in creating an appropriate spectrum of candidates for training in the source task that are also capable of being transferred to the target task.

An important consideration in transfer is whether a human can understand the knowledge being transferred among tasks. An alternative method to recycling previously learned policies directly is to take advice from learned policies to augment decision making. This advice may take the form of geometric knowledge, causal relationships, predictions, or any other type of information, allowing researchers to more easily interpret the transferred knowledge. *Rule extraction* is one such method that takes knowledge learned from a source domain and translates it into advice that aids a policy in a target domain [TTM08]. The advice is generated as a conditional, if-then statement. Torrey et al. [TTM08] describe two methods for generating advice. One method is to compose rules by decomposing the policy learned on a source task. For example, Q -values can be examined directly and rules can be generated based on which actions are preferred. An alternative method for generating advice is to analyze the *behavior* (instead of the policy) of an agent to generate rules. Consider observing agents playing a game of Keepaway soccer. Through observation, it may be apparent that a learned policy always passes the ball if opponents approach within one meter, which may then be transformed into a rule to transfer to another task. These sets of rules have the advantage of being understandable to humans, allowing researchers to know what knowledge is being transferred and how it is contributing.

Interestingly, transfer learning does not always require a designated source and target task. Instead, knowledge may transfer among several tasks that are simultaneously being learned. By encoding the knowledge for multiple tasks within the same policy, the knowledge gained from each individual task may combine with and complement the knowledge from other tasks. For example, Collobert and Weston [CW08] demonstrate transfer learning through multi-task training for natural language processing (NLP) with deep neural networks. There are many tasks related to NLP, including part-of-speech tagging, chunking, named entity recognition, semantic role labeling, language modeling, and relating words syntactically. The idea is that learning about one such task may contribute to learning the others. By training the policies simultaneously for all these capabilities, knowledge can be continually passed back and forth among all these tasks. In particular, [CW08] show that this method improves generalization and achieves competitive results in the task of relating words with similar meaning.

The approach in this dissertation focuses on the role of *representation* in task transfer. The next section describes the RoboCup domain, which is a complex machine learning domain with simpler subtasks.

2.6 RoboCup Soccer

World Cup Robot Soccer (RoboCup) was introduced as a grand challenge for AI and robotics. The overall goal of RoboCup is to generate a team that can compete with the human World Cup champions by 2050 [KAK97]. To accomplish this goal, a number of technologies must advance, including robotics design, agent architectures, sensor fusion, strategic planning [KGB05, WKM05], and multi-agent systems [KSV05, SMS06], among others. The domain complexity not only stems from the number of technologies that need to be implemented, but also from the number of subtasks that are part of soccer. For example, low level behaviors, such as moving and kicking, must be learned in conjunction with higher level subtasks, such as defending the goal and maintaining possession of the ball. The competition is held each year so that researchers can demonstrate the effectiveness of their designs against each other. There are several robot leagues, such as small, medium, and humanoid robots, in addition to the standard platform and the simulation leagues. In this way, researchers can focus on system design and not hardware differences. The remainder of this section focuses on the simulation league and the popular machine learning benchmark RoboCup Keepaway.

The RoboCup simulation league is based on the RoboCup Soccer server, a software implementation of the game of soccer that simulates many of the features that make soccer a challenging machine learning task [NNM98]. The server interacts with agents through a client/server architecture, where agents (clients) connect to the soccer simulator (server) through network communications. Agents then receive updates from the server for sensory information and send commands for actions to perform within the simulation.

Sensory information is divided into visual, auditory, and proprioceptive information. Visual information is received in the form of a heading and distance for each object seen, as well as information related to the object identity. Auditory sensing depends on the listening capacity of agents and the distance from the source, such that agents receive a heading and message for audio heard. Both visual and auditory sensing can be noisy, with information about distance and heading increasing in variance as object distance increases. Proprioception conveys changes in the agent’s physical state including information such as stamina, speed, or actions performed. Agents face the challenge of fusing this sensory information into an accurate model of the simulated world and accounting for the associated noise.

Actions are sent to the server through a string with the command identifier plus the appropriate parameters, such as heading and power for a kick. These commands are then actuated in the simulation by the server. There is also noise in the actuation, so the agents must adjust their actions and behavior in response.

Teams consist of 11 simulated agents that must each be connected to the server. Each agent is independent of the others, thus they face the challenges of multi-agent learning. The complexity of the full RoboCup soccer game poses a serious challenge to machine learning. However, one possibility is that learning methods can be applied to simpler subtasks of soccer, such as Keepaway, and then scaled up to the full RoboCup soccer task.

RoboCup simulated soccer Keepaway [SSS01] is a popular RL performance benchmark and can itself be scaled to different numbers of agents to create new versions of the same task. RoboCup Keepaway is a popular investigation tool for methods, such as neuroevolu-

tion [MEK07], temporal difference learning [SSK05], and transfer learning [TSL07], in part because it includes many of the challenges of full RoboCup, such as a large state space, partially observable state, and noisy sensors and actuators, while simplifying the domain to be more tractable for learning methods. It is also a subtask of full RoboCup Soccer, creating a stepping stone from which methods can potentially scale [KAK97, Mac09].

In Keepaway, *keepers* try to maintain possession of the ball within a fixed region and *takers* attempt to take it away. The number of agents and size of the field can be varied to make the task more or less difficult: The smaller the field and the more players in the game, the harder it becomes. A standard Keepaway benchmark setup [SSK05] is three keepers versus two takers on a 20m×20m field. In this setup, agents’ sensors are noisy and their actions are nondeterministic. Takers follow static policies, wherein the first two takers move towards the ball and additional takers attempt to block open keepers. The learner only controls the keeper who possesses the ball; its choices are to hold the ball or pass to a specific teammate.

In the 3 vs. 2 task, 13 variables represent the agent’s state [SSK05]. These include each player’s distance to the center of the field, the distance from the keeper with the ball to each other player, the distance from each other keeper to the closest taker, and the minimum angle between the other keepers and the takers (figure 2.3). The three possible actions are holding the ball or passing to one of the other two keepers. However, this dissertation explores the idea that this representation is not the only one possible for states or actions in Keepaway and related tasks, and alternative representations may enhance learning capabilities.

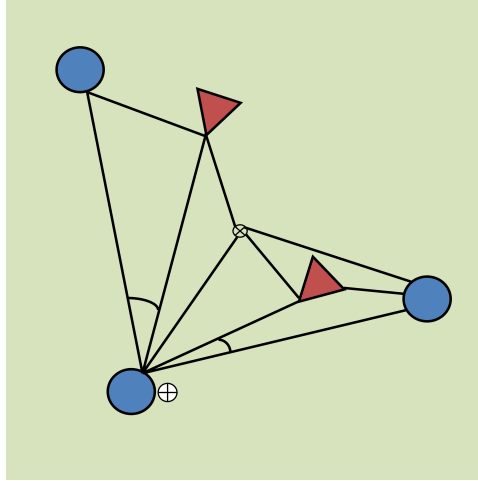


Figure 2.3: Visualization of Traditional State Variables in 3 vs. 2 Keepaway. The 13 state parameters that represent the state in the 3 vs. 2 Keepaway task are depicted in this figure. The three keepers are represented by the circles and the takers are represented by the triangles. The state parameters include the distances from each player to the center of the field (marked by the circle with the \times), the distances from the keeper with the ball (denoted by the circle with the $+$) to each other player, the distance from each other keeper to the taker nearest them, and the angles along the passing lanes. This dissertation explores alternatives to this traditional representation.

2.7 Background Summary

Representation is a critical feature in learning, but the types of representation possible are constrained by the encoding of the solution. Topology and weight evolving neuroevolution approaches optimize ANNs for a particular domain. However, directly-encoded ANNs, such as in NEAT, are constrained in size because larger representations lead to intractable dimensionality as the search space grows. Alternatively, indirect encodings may generate large structures and still be trained effectively. HyperNEAT extends NEAT to evolve an indirect encoding of ANNs.

Task transfer means applying knowledge learned in one task to a new, related task [Car97, TS07a, TSL07]. While there are many methods of task transfer, each faces common challenges: First, compatibility among tasks must be verified. Second, knowledge must be transferred from the source task to the target task. Finally, transfer must enhance, not hinder, performance in the target task. The success of transfer ultimately depends on the representation of solutions in specific tasks.

One such task, RoboCup, is a challenging machine learning domain, posing several difficulties and obstacles that must be addressed for any approach to be effective. The complexity of full RoboCup soccer is such that directly learning the task may be intractable. Instead, learning from simpler subtasks, such as Keepaway, may allow methods to scale up to the full soccer domain through task transfer. A combination of indirect encoding and state representations that enable enhanced task transfer are described next.

CHAPTER 3

APPROACH: BIRD’S EYE VIEW

A major challenge for the state representation in RL tasks is that specific state variables are often tied to agents or individual objects, which makes it difficult to add more such objects without expanding the state space (Section 2.6; [TSL07]). To address this problem, this section proposes a static representation, the bird’s eye view (BEV) perspective, which enables scaling to higher complexity states without the need to alter the representation. The BEV is explained first, followed by its implementation, which is based on the HyperNEAT approach.

3.1 Bird’s Eye View

Humans often visualize data from a BEV. Examples include maps for navigation, construction blue prints, and sports play books. Key to these representations is that they remain the same (i.e. they are *static*) no matter how many objects are represented on them. For example, a city map does not change size or format when new buildings are constructed or new roads are created. Additionally, the physical geometry of such representations allow agents to understand spatial relationships among objects in the environment by placing them in the context of physical space. The BEV also implicitly represents its borders by excluding

space outside them from its field of view. As suggested in Kuipers’ Spatial Semantic Hierarchy (SSH), such *metrical* representation of the geometry of large-scale space is a critical component of human spatial reasoning [Kui00].

A distinctive feature of the proposed representation is that not only is the agent state represented from a BEV, but it *also* requests *actions* within the same BEV perspective. For example, to request a pass the agent can indicate its target by simply highlighting it on a two-dimensional output array. That way, instead of making decisions blind to the geometry of physical space, it can be taken into account.

Egocentric data (figure 3.1a) can be mapped to an equivalent BEV by translating from local (relative) coordinates to global coordinates established by static points of reference (i.e. fiducials). The global coordinates mark the location of objects in the BEV (figure 3.1b). This translation allows mapping an arbitrary number of objects into the static representation of the BEV.

Importantly, the continuous coordinate system must be discretized so that each variable in the state representation corresponds to a single discrete location. This discretization allows the two-dimensional field to be represented with a finite set of parameters. The values of these parameters denote objects in their respective regions.

Note that while the division of the field in figure 3.1b appears reminiscent of *tile coding* [Sut96], that appearance is superficial because (1) a tile coding of the state variables in figure 3.1a would still be egocentric whereas the BEV is not, and (2) tile coding breaks the state representation into chunks that can be optimized separately whereas the HyperNEAT

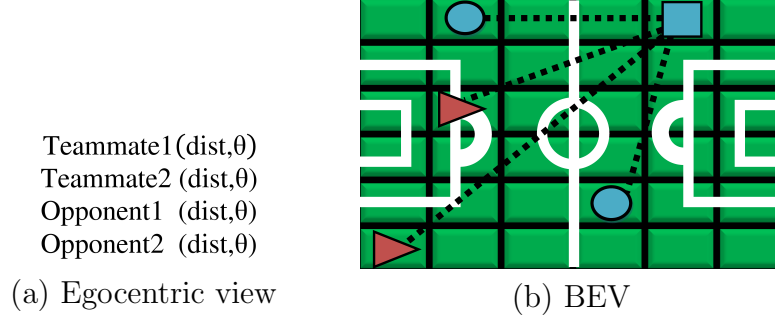


Figure 3.1: Alternative Representations of a Soccer Field. Several parameters (a) represent the agent’s relationship with other agents on a soccer field (taken from a standard RoboCup representation; [CDF03]). Each distance and angle pair represents a specific relationship of the agent to another agent. The BEV (b) represents the same relationships as paths in the geometric space. A square depicts the agent, circles depict its teammates, and triangles its opponents. The overhead perspective also makes it possible to represent any number of agents without changing the representation.

CPPN derives the connectivity of the policy network directly from the geometric relationships among the squares in figure 3.1b, as explained next.

3.2 HyperNEAT: Learning from the BEV

Geometric patterns often exhibit spatial regularities. Examples include repetition and symmetry. Furthermore, important geometric relationships such as locality and topological connectedness often critically influence informed spatial decision-making. The challenge for machine learning is that learning is often blind to the geometry of the problem, making it difficult to exploit such relationships [GS08, GS10a]. To understand the impact of learning from the true geometry of the domain, consider a two-dimensional field converted to

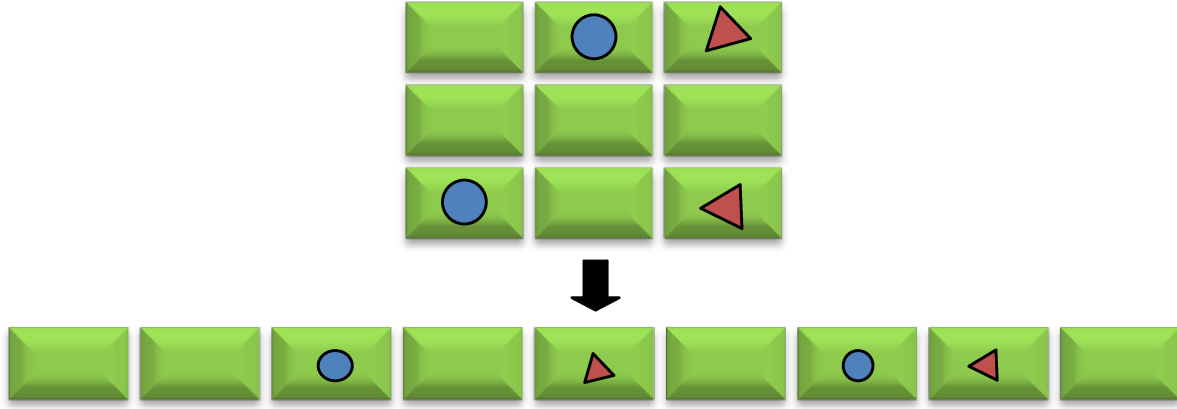


Figure 3.2: The Importance of True Geometry. A two-dimensional field transformed into a vector of parameters without any geometry forfeits knowledge of the geometry of the domain.

a traditional vector of parameters, which removes the geometry (figure 3.2). For example, consider a set of input values to an ANN such as in to figure 3.1a. Though each *dist* and θ pair is critically related in such a traditional representation, the traditional ANN has no inherent knowledge or explicit access to this relationship. In contrast, HyperNEAT *sees* the task geometry, thereby exploiting geometric regularities and relationships, such as locality, which the BEV naturally makes explicit.

For HyperNEAT to exploit patterns in a two-dimensional BEV (e.g. in soccer), the geometry of the input layer of the substrate is made two-dimensional, as in figure 3.3. That way, CPPNs can compute the connectivity of the substrate as a function of that geometry. The x and y coordinates of each input unit are in the range $[-1, 1]$. Furthermore, the output layer of the substrate matches the dimensions of the BEV so that the CPPN can exploit the geometric relationship between the input space and output space as well (figure 3.3). That the *outputs* are themselves a discretized two-dimensional plane is another significant

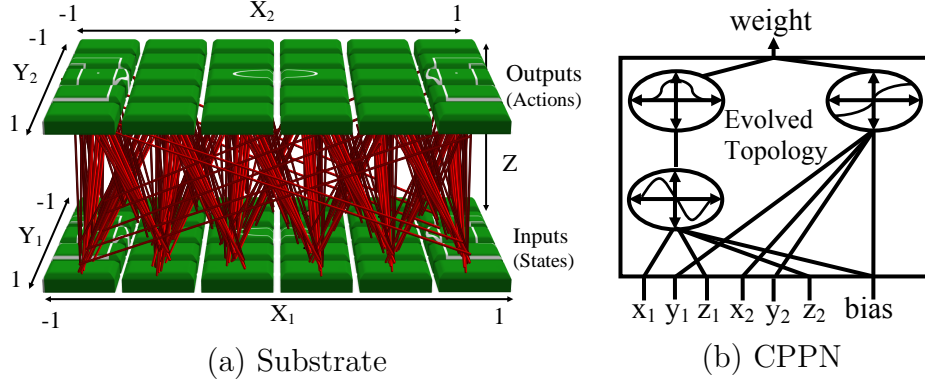


Figure 3.3: BEV Implemented in the Substrate. Each dimension ranges between $[-1, 1]$ and the input and output planes of the substrate (a) are equivalently constructed to take advantage of geometric regularities between states and actions. Because CPPNs (b) are an indirect encoding, the high dimensionality of the weights does not affect performance. (The CPPN is the search space.)

difference from tile coding. Each (x, y) -coordinate in this substrate represents a discretized region of the overhead view of physical space. Finally, the input and output layers are positioned on opposite ends of the z -axis. A six-dimensional CPPN with inputs x_1, y_1, z_1, x_2, y_2 , and z_2 determines the weights between coordinates in the two-dimensional input layer and the two-dimensional output layer, creating a pattern of connections between regions in the physical space.

To represent world state, objects and agents are literally “drawn” onto the input substrate, which is a static size, like marking a map. The generated network then can make decisions based on the relationships of such features in physical space and thereby learn the significance of certain kinds of geometric relationships among objects that are not identified a priori by the designer.

In this way, the BEV makes it possible to add new features (e.g. a new player) to the

state space *without* the need to add new inputs. Instead, they can now simply be drawn onto the existing representation with no additional apparatus. That way, task transfer to different numbers of players is made simple through the static representation.

Interestingly, although the BEV is naturally held static its size or resolution can be changed *without* retraining. A unique feature of CPPNs (which encode the BEV connectivity) is that the same CPPN can query substrates of arbitrary size or resolution. It is important to note that even when size or resolution are changed, the *CPPN* itself remains the same. Thus the BEV can extend its representation to different field sizes or to different levels of detail (i.e. resolutions), as shown in figure 3.4. In this way, the CPPN allows not only transfer to different numbers of players, but to different field sizes and resolutions, all without the need for retraining.

It is important to understand that the dimensionality of the search space in HyperNEAT is not the same as the dimensionality of the substrate because the search space is the *CPPN*, which is a compact encoding of the pattern of connections in the substrate. For example, if the substrate resolution is 20×20 then the number of possible connections in the substrate is $400 \times 400 = 160,000$. However, a CPPN that encodes this connectivity can itself contain orders of magnitude fewer connections. This fact also explains why resolution can increase without retraining. For example, if resolution increases to 40×40 (2,560,000 possible connections), there are new connections that connect locations that previously did not exist at 20×20 . However, the same CPPN can simply query the $(x_1, y_1, z_1, x_2, y_2, z_2)$ coordinate of the new connections, thereby interpolating the weights of the new connections

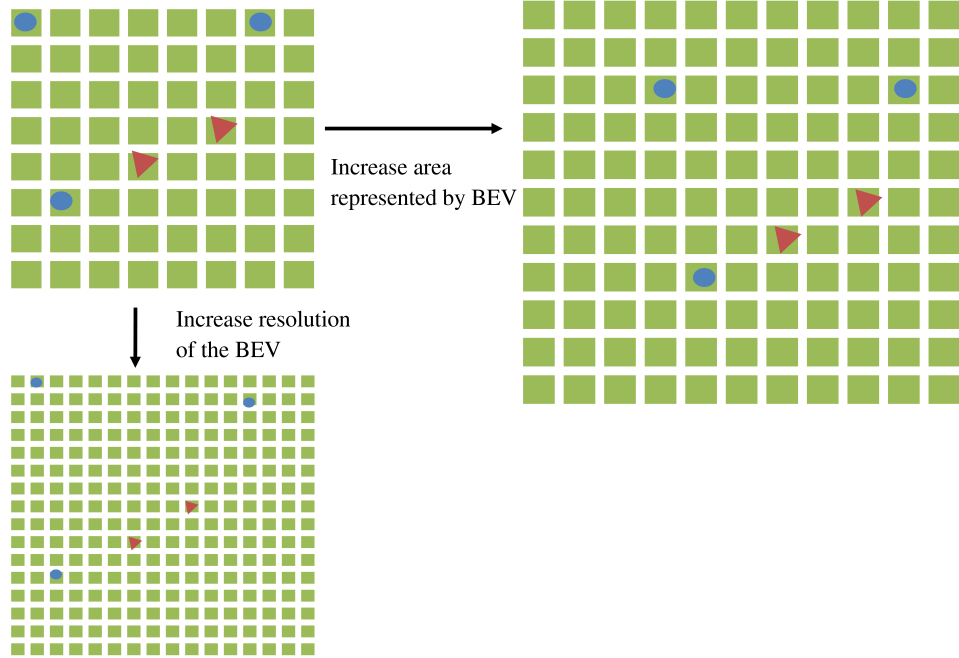


Figure 3.4: Changing the BEV. Two kinds of alterations are depicted in this figure. First, the BEV can be altered by increasing the *area* of the substrate while maintaining the size of each discrete cell by extrapolating new connection weights associated with previously unseen cells. Second, resolution is increased by increasing the number of cells and shrinking the area represented by each discrete cell. The CPPN automatically interpolates connection weights for the new locations. This way, the BEV allows new forms of transfer to differing field sizes or levels of precision.

automatically. Although the number of connections in this example increases from 160,000 to 2,560,000, the dimensionality of the CPPN does not change.

The next chapter presents experiments that demonstrate the benefits of this approach.

CHAPTER 4

ROBOCUP KEEPAWAY BENCHMARK EXPERIMENTS

The experiments in this chapter, published in the *Journal of Machine Learning Research* (JMLR) [VS10a] and the *Proceedings of the Genetic and Evolutionary Computation Conference* [VS10b], demonstrate the power of the BEV in the popular RoboCup Keepaway machine learning benchmark. The goals of the investigation are two-fold. First, the BEV establishes that it can learn in this popular domain. It is evaluated in a standard benchmark setup to which multiple methods have been applied. Results establish that the BEV is not only capable of learning the task, but also that it is competitive with traditional approaches.

Second, the BEV is investigated for its advantages. In particular, the study focuses on the ability to transfer knowledge. This investigation is performed through two different variations of transfer. One investigation varies the number of agents in the Keepaway task. Increasing the number of agents in the domain alters the difficulty of the task; however, knowledge from the simpler task with fewer agents should provide an advantage. The other investigation is transfer between tasks with different semantics. This transfer is evaluated by training on the Knight Joust domain and transferring to 3 vs. 2 Keepaway. While both domains take place in a two-dimensional plane, they otherwise do not correspond.

4.1 RoboCup Keepaway Benchmark

Because the RoboCup Keepaway domain is a popular and challenging benchmark for machine learning methods, it is chosen to demonstrate the capabilities of the BEV. The standard Keepaway benchmark setup [SSK05] is three keepers versus two takers on a $20\text{m} \times 20\text{m}$ field. To investigate the ability of a static representation, i.e. the HyperNEAT BEV, to learn this task, it is compared to both static policies [SKT06] and the learning algorithms Sarsa [RN94], NEAT [SM04], and EANT [MEK07]. Unlike the BEV, the traditional representation (with 13 state variables) through which these methods learn in 3 vs. 2 Keepaway must be changed for different versions of the task, such as 4 vs. 3 Keepaway. The static benchmarks are Always-Hold, Random, and a Hand-Coded policy, which holds the ball if no takers are within 10 meters [SS01]. These static benchmarks provide a baseline to validate that the BEV learns a non-trivial policy in the initial task.

4.1.1 *Approaches Compared*

This section describes learning approaches compared to the HyperNEAT BEV in the Keepaway domain. State action reward state action (Sarsa; [RN94]) is an on-policy temporal difference reinforcement-learning method that learns the action-value function $Q(s, a)$. The quintuple (s, a, r, s', a') defines the update function for $Q(s, a)$ by determining for a current

state (s) and action (a) what the reward (r) and the expected reward for the predicted next state (s') and action (a') will be. The update equation is

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma Q(s', a')),$$

where α is the learning rate and γ is the discount factor for the future reward. The values in $Q(s, a)$ determine which action is taken in a given state by selecting the maximal value. Each keeper separately learns which action to take in a given state to maximize the reward it receives [TWS06].

Regular NEAT [SM02] evolves ANNs to maximize a fitness function. The ANN receives the 13 state inputs (like Sarsa) to define the state of the system and produce three outputs to select an action. The fitness in RoboCup Keepaway is the average length of time that keepers can hold the ball over a number of trials [TWS06]. EANT [MEK07] is an additional neuroevolution algorithm based on NEAT that learned Keepaway. Though similar to NEAT, it distinguishes itself by more explicitly controlling the ratio of exploration to exploitation during the evolutionary process. These methods were chosen for comparison because they have been tested in the same Keepaway configuration.

4.1.2 *Experimental Setup*

As described in Chapter 3, the HyperNEAT BEV transforms the traditional state representation explicitly to capture the geometry. The standard substrate is a two-dimensional 20×20 input layer connected to a 20×20 output layer. Thus both the state and action spaces have 400 dimensions each ($p_1 \dots p_{400}$ and $a_1 \dots a_{400}$). As with Sarsa in Stone et al. [SS01], this policy representation does not include a hidden layer. However, the CPPN that encodes its weights *does* evolve internal nodes.

Each node in a substrate layer represents a 1m^2 discrete chunk of Keepaway field. Each keeper’s position is marked on the input layer with a positive value of 1.0 in its containing node and takers are similarly denoted by -1.0 . Paths are literally drawn from the keeper with the ball to the other players (as in figure 4.1).

Positive values of 0.3 depict paths to other keepers and values of -0.3 depict paths to takers. These input values for agents and paths are experimentally determined and robust to minor variation. Actions are selected from among the output nodes (top layer of figure 3.3) that correspond to where the keepers are located: If the highest output is the node where the keeper with the ball is located, it holds the ball. Otherwise, it passes to the teammate with the highest output at its node. This method of action selection thus corresponds exactly to the three actions available to Sarsa, NEAT, and EANT. A key property of this representation is that it is independent of the number of players on either side, unlike the representation in the traditional approaches.

HyperNEAT evolves the CPPN that encodes the connectivity between the ANN layers

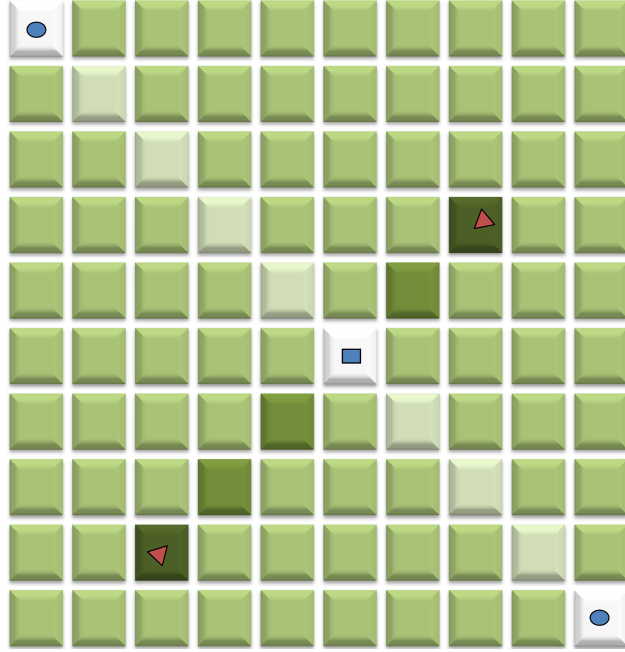


Figure 4.1: Visualizing the BEV Input Layer in 3 vs. 2 Keepaway. The BEV input layer is marked with the positions of keepers, takers and paths. The keeper with the ball is the small square, other keepers are circles, and the takers are triangles. Positive values are denoted by lighter shades (for keepers and paths to keepers) and negative values are denoted by darker shades (for takers and paths to takers). The middle shade represents an input of 0.0, the lightest is +1.0, and the darkest is -1.0. The BEV represents the distances and angles in a geometric configuration, allowing geometric relationships to be exploited by HyperNEAT. Paths represent ball position by converging on that keeper.

in the substrate (up to 160,000 connections with a 20×20 resolution). Fitness is assigned according to the generated network’s ball possession time averaged over at least 30 trials, with additional trials up to 100 assigned to those above the mean, following Taylor et al. [TWS06]. Additionally, the CPPNs in the initial population are given the geometric concept of locality (Section 2.4). This experiment is performed in the original RoboCup Soccer Server simulator. Parameters for this experiment are in Appendix B.

4.1.3 Results

In the RoboCup Keepaway benchmark, performance is measured as the number of seconds that the keepers maintain possession [SS01, SKT06, TWS07]. After training, the champion of each epoch is tested over 1,000 trials. Performance results are averaged over five runs with each consisting of 50 generations of evolution. This number of generations was selected because the corresponding *simulated* time spent in RoboCup during training equals the simulated time (800-1,000 hours) for previous approaches (Sarsa, NEAT, and EANT; [TWS06, MEK07]). The test on the 3 vs. 2 benchmark is intended to validate that the BEV learns competitively with other leading methods.

In 3 vs. 2 Keepaway on the 20m×20m field, the best keepers from each of the five runs controlled by a BEV substrate trained by HyperNEAT maintain possession of the ball on average for 15.4 seconds ($sd = 1.31$), which significantly outperforms ($p < 0.05$) all static benchmarks (Table 4.1). Furthermore, assuming similar variance, this performance significantly exceeds ($p < 0.05$) the current best reported average results [SSS01, SSK05, TWS06] on this task for both temporal difference learning (12.5 seconds) and NEAT (14.0 seconds), and matches EANT (14.9 seconds; Table 4.1). The important implication of this result is that the HyperNEAT BEV is competitive with the top learning algorithms on this task.

Table 4.1: Average Best Performance by Method. The HyperNEAT BEV holds the ball longer than previously reported best results for neuroevolution and temporal difference learning methods. Results are shown for Evolutionary Acquisition of Neural Topologies (EANT) from Metzen et al. [MEK07], NeuroEvolution of Augmenting Topologies (NEAT) from Taylor et al. [TWS06], and State action reward state action (Sarsa) from Stone and Sutton [SS01].

Method	Average Hold Time
HyperNEAT BEV	15.4s
EANT	14.9s
NEAT	14.0s
Sarsa	12.5s
Hand-tuned Benchmark	8.3s
Always Hold Benchmark	7.5s
Random Benchmark	3.4s

4.2 Task Transfer

To evaluate its advantage in task transfer, the BEV is evaluated in several forms of transfer. Transfer from the 3 vs. 2 Keepaway task to a 4 vs. 3 Keepaway task is evaluated first. These additional agents increase both the complexity and difficulty of the task. Second, transfer from a significantly different domain is evaluated through transfer from the simple grid world domain of Knight Joust to 3 vs. 2 Keepaway. All transfer experiments are in the RoboCup Soccer Sever simulator.

4.2.1 Keepaway 3 vs. 2 to 4 vs. 3

Transfer between Keepaway tasks is evaluated by training policies on 3 vs. 2 and then testing on *both* the 3 vs. 2 and 4 vs. 3 tasks for 1,000 trials each without any further training. Note that this evaluation of transfer differs from Taylor et al. [TWS07], in which teams trained in the smaller task are *further trained* in the larger task after the transfer because new parameters are added. In contrast, transfer within the BEV requires no changes or transformations. Performance is averaged over five runs, following Taylor et al. [TWS06]. Figure 4.2 shows the average test performance on *both* 3 vs. 2 (trained) and 4 vs. 3 (untrained) of each generation champion.

Testing performance in the 3 vs. 2 task improves to 14.3 seconds on average over each run. At the same time, the test untrained performance of these same individuals in the 4 vs. 3 task, which was not trained, improves from 6.6 seconds to 8.1 seconds on average. In contrast, the previous best approach to transfer learning in this domain required executing a transfer functional and additional training for between 50 and 200 hours (depending on the chosen transfer function) *beyond* the initial bootstrap training in 3 vs. 2 to achieve a comparable 8.0 second episode duration [TWS07]. Thus because the BEV is static, transfer improves and requires no special adjustments to the representation to achieve the same result as many hours of *further* training with the TVITM-PS transfer method.

Although the BEV improves in 4 vs. 3 Keepaway even when only trained in 3 vs. 2, it is still informative to investigate the effect of further training in the 4 vs. 3 task on asymptotic performance. For this purpose, individuals are trained in the 3 vs. 2 task for 20 generations

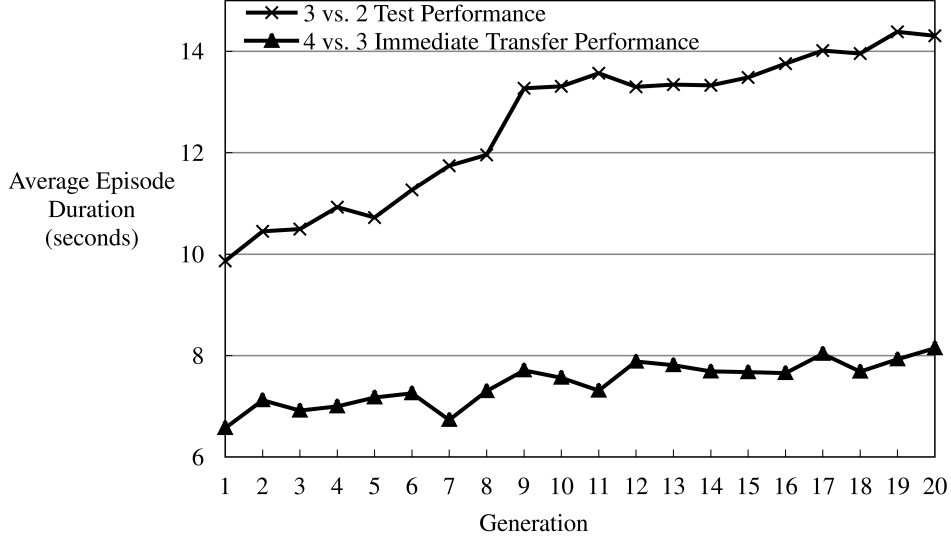


Figure 4.2: Transfer Learning From 3 vs. 2 to 4 vs. 3 Keepaway on a 25m×25m Field. As the performance (averaged over five runs) of the champion in the 3 vs. 2 task improves, the untrained performance in the 4 vs. 3 task also consequently improves from 6.6 seconds to 8.1 seconds without *ever* training for it. The improvement is positively correlated ($r = 0.87$).

and then further trained in the 4 vs. 3 task for 30 generations. The asymptotic performance of these policies is contrasted with keepers trained on 4 vs. 3 from scratch for 50 generations. Performance is averaged over five runs and generation champions are evaluated over 1,000 episodes. Figure 4.3 shows the average test performance of the generation champions. The individuals trained solely on 4 vs. 3 improve from 6.2 seconds to 8.0 seconds. Interestingly, this performance is equivalent to policies trained only in the 3 vs. 2 task and transferred to 4 vs. 3. However, individuals trained on 3 vs. 2 for the first 20 generations increase their test performance on 4 vs. 3 to 9.1 seconds over the last 30 generations. The final difference between further training after transfer and training from scratch is significant ($p < 0.05$).

An important factor in the superior performance of the learner that was transferred is

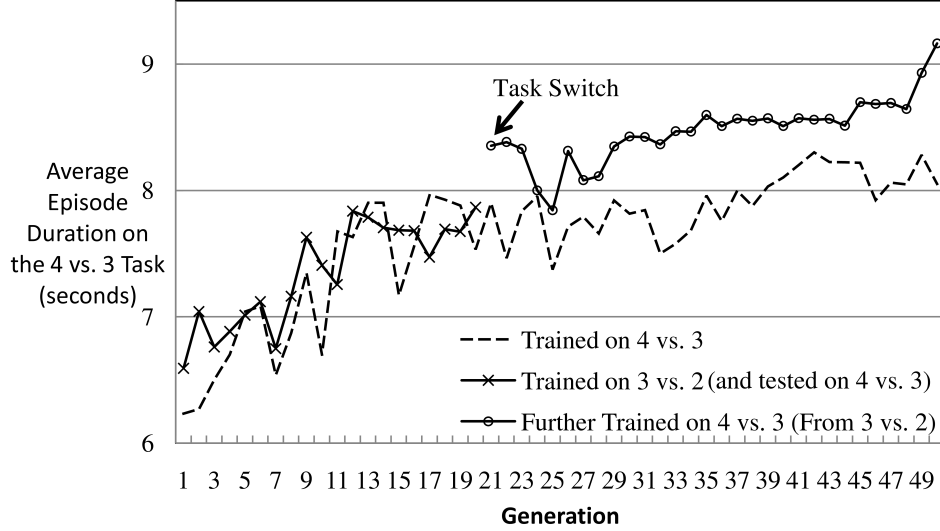


Figure 4.3: Further Training After Transfer From 3 vs. 2 to 4 vs. 3 Keepaway on a $25m \times 25m$ Field. Performance of individuals trained on 3 vs. 2 then transferred to 4 vs. 3 and further trained are contrasted with individuals solely trained on 4 vs. 3. All depicted results are performance in the 4 vs. 3 task. Prior training on 3 vs. 2 and transfer to the 4 vs. 3 enhances keeper performance by beginning in a more optimal area of the search space.

the behavior of the third taker in the 4 vs. 3 task, which seeks to block the most open player. This behavior differs from 3 vs. 2, in which the two takers attempt to take the ball only by always heading towards it. When training on 4 vs. 3 without previously learning on 3 vs. 2, the third taker's behavior may inhibit performance by preventing important knowledge from being learned. For example, in 3 vs. 2 an important concept is to pass to the most open player. However, in 4 vs. 3 the most open player is *not* always the best choice because of the behavior of the third taker; therefore, policies that learn the concept of passing to the most open player, which is still an important skill, are not discovered.

4.2.2 *Knight Joust to Keepaway*

Knight Joust is a predator-prey variant domain wherein the player (prey) starts on one side of the field and the opponent (predator) starts on the opposite side [TWS07]. The player must then travel to the opposite side of the field while evading the opponent. The name *Knight Joust* reflects that the player is allowed three potential moves: move forward, knight jump left, and knight jump right, where a *knight jump* is two steps in the direction left or right and then forward (as in chess). The opponent follows a stochastic policy that attempts to intercept the player. The traditional state representation consists of the distance to the opponent, the angle between the opponent and the left side, and the angle between the opponent and the right side (figure 4.4).

While Knight Joust is significantly different from Keepaway, a feature of both is that at each step the agent must make the decision that best avoids the opponent. However, Knight Joust is simpler, eliminating such complexity as multiple agents, noise, and kicking a ball, making it more tractable. The simplification makes it ideal for cross-domain transfer; because training is quicker and easier than in Keepaway, knowledge is more quickly bootstrapped. In Taylor et al. [TSL07], cross-domain transfer from Knight Joust to Keepaway was shown to enhance learning. Additionally, the HyperNEAT BEV can represent the state information in figure 4.4 by drawing the state information onto the inputs.

Cross-domain transfer is evaluated from the task of Knight Joust on a 20×20 grid to 3 vs. 2 Keepaway on a $20m \times 20m$ field. Evolution is run for 20 generations in the Knight Joust task and then the champions seed the beginning generations of 3 vs. 2 Keepaway. Further

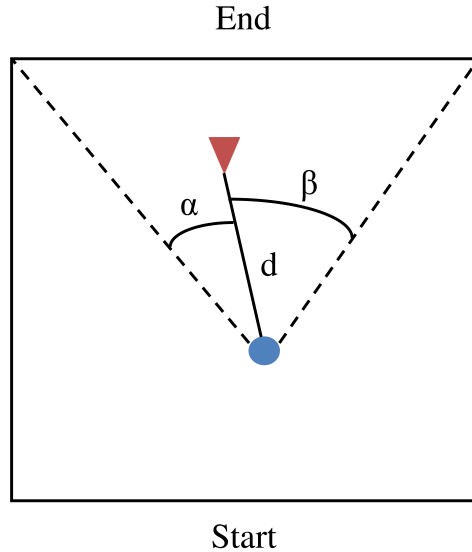


Figure 4.4: Knight Joust World. In Knight Joust, the player (circle) begins on the side marked *Start* and must reach the side marked *End*, while evading the opponent (triangle). The player is given the state information of the distance to the opponent, d , the angle between the opponent and the left side, α , and the angle between the opponent and the right side, β . This state information can similarly be drawn on the substrate of the BEV by marking the position of the player, opponent, the path between them, and the paths to the corners.

training is then performed over ten additional generations of evolution. Jumpstart performance in Keepaway of the champion players from Knight Joust is on average 0.3 seconds above the performance of initial random individuals. After one generation of evolution, the best individuals from transfer exceed the raw performance by 0.6 seconds. Finally, after ten further generations, the best individuals with transfer hold the ball for 1.1 seconds longer than without transfer (figure 4.5). Thus the total reward for the keepers with transfer is greater than the total reward without transfer.

These differences are significant ($p < 0.05$). Thus even preliminary learning in a significantly different domain proved beneficial to the BEV. In contrast, previous transfer results

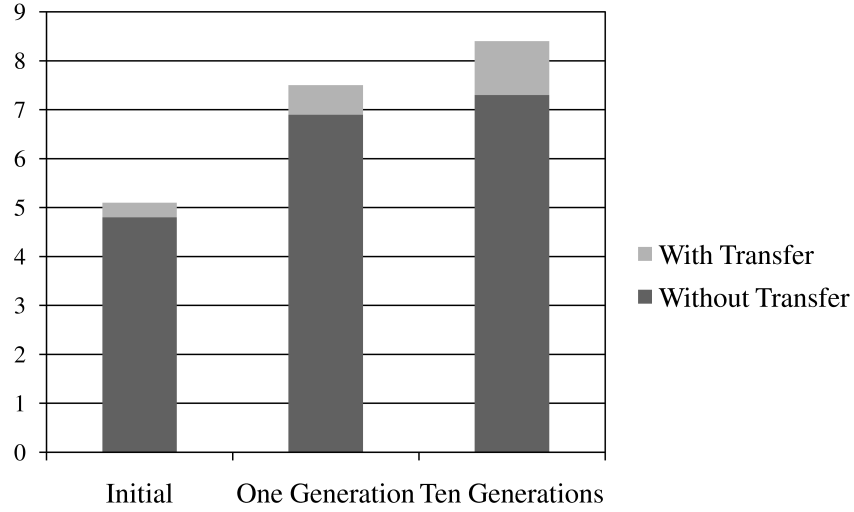


Figure 4.5: Transfer Results from Knight Joust to Keepaway. Untrained and further training performance averaged over 30 runs is shown. The performance of raw champions from Knight Joust on Keepaway exceeds initial random individual by 0.3 seconds. After one generation, this advantage from transfer increases to 0.6 seconds and at 10 generations the advantage is 1.1 seconds. Thus performance on Keepaway, both instantaneous and with further training, benefits from transfer from the Knight Joust domain with significance $p < 0.05$.

from Knight Joust to Keepaway from Taylor et al. [TS07b] demonstrated an initial performance advantage, but after training for five simulator hours (which is less than the duration of ten generations) there was no performance difference between learning with transfer and without it.

4.3 Summary

In summary, the BEV simplifies task transfer by making the state representation static. That way, no matter how many objects are in the domain, the size of the state representation remains the same. In contrast, in traditional representations, changing the number of players (e.g. in the RoboCup Keepaway task) forces changes in the representation by adding dimensions to the state space. In addition to results competitive with leading methods in the Keepaway benchmark, the BEV, which is enabled by an indirect encoding, achieved transfer learning from 3 vs. 2 to 4 vs. 3 Keepaway *without* further training and cross-domain transfer from Knight Joust to Keepaway.

CHAPTER 5

SUBSTRATE CONFIGURATION EXPERIMENTS

As ANNs become increasingly large and complex, such as those that underlie the BEV, the question of how to organize and constrain such structures becomes increasingly important. Organizational principles are key to effectively optimizing such complex structures. In prior experiments, the BEV contained a large number of inputs, outputs, and connections and thus results in a high-dimensional structure. One avenue to create effective high-dimensional ANNs is constraining connectivity. In particular, modularity requires limiting connectivity among independent modules. Furthermore, though high-dimensional, the BEV so far has contained no hidden nodes. Thus the decision function determined by the BEV is in effect linear. Hidden nodes can allow the BEV to compute nonlinear decisions, but the question for any particular domain is whether the BEV needs to make a nonlinear decision.

This chapter accordingly details experiments that explore the organization and configuration of the ANN substrate.

5.1 Modularity

As tasks increase in complexity and difficulty, larger and more complex structures are required to solve them. Organizational principles are key to effectively training such complex

structures. Natural structures exhibit several important organizational principles, for example modularity, regularity, and hierarchy [HHL99, Lip07, WA96], which enable evolution to scale structures in complexity and size. HyperNEAT captures the principle of regularity, but the question remains what other organizing principles may be captured by the HyperNEAT approach. This section focuses in particular on how modularity can be encouraged as a *function of geometry* through the indirect encoding in HyperNEAT.

The challenge in encouraging modularity is to bias search towards modular solutions without restricting the search space to modularity. The danger in explicitly designing modularity into the representation is that it may bias search towards modular structure when non-modular designs may be required, inhibiting search from finding an optimal solution [GGW04]. For example, fully-connected artificial neural networks (ANNs) are completely non-modular. To produce a modular ANN, connectivity would be restricted, but that could prevent solutions that may require full connectivity.

Traditional HyperNEAT has been shown to generate regularities in ANN weight patterns in tasks such as checkers [GS10a], robot control [SDG09], RoboCup Keepaway (Chapter 4), and quadruped locomotion [CSP11], but Clune et al. [CBM10] showed that it struggles to produce modular patterns. To address this problem, this section introduces a novel insight that can consistently bias HyperNEAT towards modular structures. The initial idea is that a bias towards *locality* is an important prerequisite to modularity. However, locality alone is not enough because not *all* connectivity should be local. Thus the key insight is that the search should *start* with a bias towards locality that can later be adjusted by evolution

to different levels for different parts of the network geometry. This idea is explored in a HyperNEAT variant called HyperNEAT with Link Expression Output (HyperNEAT-LEO).

To demonstrate its advantage, HyperNEAT-LEO is compared to the standard HyperNEAT method (with and without locality bias) and another variant that enforces an external distribution on connectivity. The HyperNEAT variants are investigated in three domains. First, the variants are evaluated in the Knight Joust domain with the high-dimensional bird’s eye view representation [VS10a]. The BEV has a large number of connections (160,000) between two-dimensional planes. An interesting result is that each of the variants has its own unique pattern of expressed connections but performance is not altered, indicating that for the Knight Joust task connectivity can be constrained in a number of different ways without negative impact.

The question that then arises is when constraining connectivity provides an advantage. This question is addressed in the second domain, the Retina Left and Right problem [CBM10] (based originally on Kashtan and Alon [KA05], in which the ANN must indicate whether particular objects are detected in the left and right retinas). The Retina problem is modular because the calculations for the left and right sides are independent, which has proven difficult for HyperNEAT to represent [CBM10]. In this task, a significant difference is demonstrated among the variants. The standard HyperNEAT does not perform well at the task, finding a solution only 12% of the time or less. In addition, both biasing standard HyperNEAT towards locality in *weights* and imposing a local connectivity distribution perform even worse and never find a solution. However, interestingly, HyperNEAT-LEO, which evolves the pat-

tern of expressed connections, is able to achieve up to a 91% success rate if it begins with a bias towards local connectivity. These results demonstrate that modularity can be encoded as a function of geometry, providing a means to evolving modularity in HyperNEAT.

Finally, HyperNEAT-LEO is applied to the standard RoboCup Keepaway benchmark with the BEV. In the Keepaway domain, there is no clearly-defined modularity, such as in the Retina Left and Right problem, and biasing connectivity towards locality is not an obvious advantage. Nevertheless, the performance of standard HyperNEAT is compared to HyperNEAT-LEO and the connectivity of solutions is examined. HyperNEAT-LEO with a bias towards locality significantly outperforms standard HyperNEAT in this task, improving performance by 0.89 seconds in hold time. More interesting is the connectivity of the solutions. Standard HyperNEAT’s average population and even average champions’ connectivity patterns are frequently near full connectivity. In contrast, HyperNEAT-LEO begins with highly disconnected solutions that gradually increase in connectivity over time.

5.1.1 Modularity in Evolutionary Computation

Modularity can be defined as the separability of structure into functionally independent units [CBM10, KA05, Lip07]. Modularity is common to both biological and engineered systems. For example, cells assemble into tissues, tissues into organs, and organs into organisms. Decomposing a design into modules can allow evolutionary algorithms to address com-

plex tasks that require sub-functions of the solution to be independently optimized [KA05]. Approaches to modularity in evolutionary algorithms range from encapsulating modules [GGW04, WAG09] to alternating tasks to facilitate their discovery [KA05].

A common approach to modularity in evolutionary algorithms is to capture or create modules through high-level processes. For example, *modularity-preserving representations* preserve the modularity that exists in the phenotype space by transferring it to the genotype space through encapsulation [GGW04, WAG09]. The process of encapsulation creates new genotypic primitives that are groupings of existing primitives. In this way, modules can be generated and assembled together to produce solutions. Such high-level processes for evolving modularity are dependent on the capability of the algorithm to select beneficial modules and may be detrimental to search by sometimes encapsulating modules that are not useful.

Alternatively, the evolutionary algorithm may be modified such that fitness pressure is applied to encourage discovering modularity. In *modularly varying goals* evolution [KA05], the task that is being solved alternates at set generational intervals. Bias towards modularity is created by selecting tasks in which there are shared sub-goals because it is easier to switch between tasks if the appropriate modules for these sub-goals are discovered. For example, the modularly varying goals version of the Retina Problem [KA05] switches between the tasks of (1) determining whether there is a target pattern in the right *or* left retina and (2) determining whether there is a target pattern in the right *and* left retina. These tasks must independently discover the appropriate functions for the left and right retinas, and then combine them correctly depending on the task. The challenge for setting up such a

scenario is selecting tasks that share modules such that their discovery is beneficial.

Clune et al. [CBM10] applied HyperNEAT to the modularly-varying goals Retina Problem. Standard HyperNEAT struggled to produce modularity in this task and achieved significantly fewer correct classifications than the direct encoding implemented by Kashtan and Alon [KA05]. In addition to the modularly-varying goals version of the Retina Problem, HyperNEAT was also tested in the non-varying Retina Problems, i.e. the fixed goal of right *and* left separately evolved from and the fixed goal of right *or* left, and in alternative Retina Problems that must judge each retina’s input individually. For example, a left output indicates validity for the left pattern and a right output indicates validity on the right. The most difficult of all these tasks for HyperNEAT was the Retina Left and Right task [CBM10], which must correctly judge as valid or invalid each retina’s pattern separately. Interestingly, Clune et al. [CBM10] found that *imposing* modularity by disabling connections between the left and right sides of the network improved HyperNEAT’s performance [CBM10]. This enforced modularity demonstrates that modularity could potentially provide HyperNEAT an advantage in this task, but that HyperNEAT struggles to create modularity on its own.

Experiments in this section, published in the *Proceedings of the Genetic and Evolutionary Computation Conference* [VS11], add to our understanding of modularity by focusing on the role of geometry in encoding modularity. The next section explains modifications to HyperNEAT based on this idea that alter how connection expression is controlled.

5.1.2 *HyperNEAT Thresholding and Link Expression Output*

Although the HyperNEAT method succeeds in a number of challenging tasks [GS10a, SDG09, VS10a] by exploiting geometric regularities in the solution, it is less effective at generating modular solutions [CBM10]. Instead it tends to generate fully-connected networks even when limited connectivity is beneficial. The conventional method for controlling connectivity in HyperNEAT is a threshold that limits the range of values output by the CPPN that can be expressed as weights [GS10a, SDG09] (Section 2.4). The threshold is a parameter specified at initialization that is uniformly applied to all connections queried. When the magnitude of the output of the CPPN is below this threshold, the connection is not expressed. In this way, the proportion of connections expressed may be controlled by raising or lowering the threshold when fewer or more connections are desired. Because the threshold is *uniform*, it influences every connection equally with no bias towards modularity (which likely requires different levels of connectivity in different areas).

As an alternative to the traditional uniform threshold, one idea to control connection expression is a *dynamic threshold*. The dynamic threshold varies depending on the specific nodes of the connection being queried. As each connection is queried, the threshold, which determines whether the CPPN-generated weight should be expressed as a connection, is updated and then applied. In this way, a particular expression distribution that is determined a priori can be enforced to influence the pattern of connectivity in the ANN. In particular, this section explores a dynamic threshold based on distance with the value $t = c + c * ||\vec{n}_i - \vec{n}_j||$, where t is the local threshold value, c is a constant parameter set at

initialization and $||\vec{n}_i - \vec{n}_j||$ is the Euclidean norm of the difference between the endpoint-nodes' coordinate-vectors. This threshold makes sense for modularity because it expresses the concept of *locality* by discouraging connections of greater distance. That is, the higher the distance is between nodes, the higher the threshold will be.

In another alternative to the uniform threshold, instead of thresholding connections through an external value, HyperNEAT itself is extended to generate an *expression pattern* that controls whether connections are expressed at different locations independently of their weights. For this purpose, a new Link Expression Output (LEO) is added to the CPPN that indicates whether a connection is expressed. When the LEO (which is a step function) is greater than 0.0 the connection weight is set to the usual CPPN weight output; otherwise it is not expressed (algorithm 2). The key lines that are changed from algorithm 1 are 5–8. In this way, the pattern of connectivity can be controlled by the CPPN itself *independently of the weights*, and that connectivity can be evolved as a function of geometry. This approach follows the idea that connection expression and weights are best considered separately.

Furthermore, because expression patterns are functions of geometry (as are the weights), evolution can be seeded with geometric principles that bias the pattern of connectivity. Evolution is seeded by initializing CPPNs in the first generation with seed topographies that represent important concepts. For example, locality can be expressed through a Gaussian function. Because the Gaussian function peaks when its input is 0.0, inputting a difference between coordinates, e.g. Δx , achieves the highest value when the coordinates are the same.

<pre> Input: Substrate Configuration Output: Solution CPPN 1 Initialize population of minimal CPPNs with random weights; 2 while <i>Stopping criteria is not met</i> do 3 foreach <i>CPPN in the population</i> do 4 foreach <i>Possible connection in the substrate</i> do 5 Query the CPPN for weight w of connection and expression value e; 6 if $e > 0.0$ then 7 Create connection with a weight w; 8 end 9 end 10 Run the substrate as an ANN in the task domain to ascertain fitness; 11 end 12 Reproduce CPPNs according to the NEAT method to produce the next generation; 13 end 14 Output the champion CPPN. </pre>
--

Algorithm 2: HyperNEAT-LEO Algorithm

To seed for locality on multiple dimensions, a Gaussian node can be added for each such dimension, as shown in figure 5.1. Such seeds provide the concept of locality because the more local the connection (i.e. $\Delta x \simeq 0.0$), the greater the impact of the Gaussian function. This extension is called HyperNEAT with a Link Expression Output (HyperNEAT-LEO).

Each of the three variants discussed in this section provides different means of controlling connectivity and encouraging modularity. The next section introduces the experiments designed to demonstrate the differences among these approaches to thresholding.

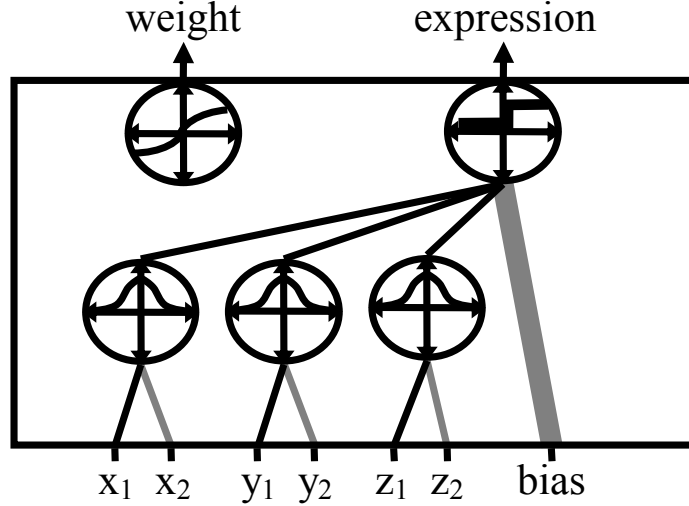


Figure 5.1: Seed CPPN Expressing Global Locality. Initial CPPN topologies imparted with geometric principles can seed evolution. For example, a six-dimensional CPPN can be initialized with three hidden nodes connected by positive (black) and negative (gray) weights. These hidden nodes take as input $x_1 - x_2$ (Δx), $y_1 - y_2$ (Δy), and $z_1 - z_2$ (Δz) respectively and have Gaussian activation functions. The nodes are connected to the LEO, which has a bias of -3 . Because the Gaussian function peaks at 0.0 , the expression output value is greater than or equal to 0.0 only when Δx , Δy , and Δz are 0.0 . Thus the initial population is seeded with CPPNs that constrain connectivity to respect locality. When seeding in this way, the weight output of the CPPN is included as usual with a set of direct connections from all the inputs (not shown).

5.1.3 Experimental Setup

The experiments in this section are designed to investigate the effect of different methods of thresholding on connectivity and modularity. The key focus is on the idea that an effective means of expressing modularity is as a *function of geometry*. This geometric function may be externally defined or evolved as a geometric pattern expressed by the CPPN. One key to modularity is limiting the number of connections, but limiting the connections alone does not necessarily lead to modularity. This section explains the task designed to explore how

potential thresholding approaches influence modularity.

The first domain, Knight Joust, is a simple grid world [TWS06], which was introduced in Section 4.2.2. In this domain, the agent must move from the starting side of the grid to the opposite side while avoiding the opponent. Each node corresponds to a grid cell in the world. Thus objects can be marked at their position in the grid world on the BEV. Because the BEV has a large number of connections (160,000) and only a small proportion of them may be needed, it is suited to an investigation into connectivity constraint. The large number of potential connections and simple substrate geometry facilitate examining and interpreting the effects of thresholding on connectivity. Each of the three variants of HyperNEAT train CPPNs in the Knight Joust task with the same parameters except for how thresholding is performed. The *uniform threshold* has a value of 0.3 and the *dynamic distance threshold* has a constant of 0.15. *HyperNEAT-LEO* has no additional parameters because its additional CPPN output node replaces the traditional threshold. The connectivity of the final solutions from each of 100 runs is examined. The aim of the experiment with Knight Joust is to understand how different thresholding techniques impact *locality*. To investigate their impact on modularity in particular, a second task that is known to be modular will be detailed shortly.

Introduced by Kashtan and Alon [KA05], the Retina Problem, which is the inspiration for the second domain, poses the challenge of creating an ANN that must identify patterns input into the left and right retinas. The retinas consist of four inputs each that are each set to one of two values. Because the values for each input are binary, a four-input retina

can take 16 possible patterns and there are overall 256 configurations for two four-input retinas. The individual retinas each have a unique set of specified patterns identified as valid or invalid that are equally divided into eight patterns each.

In the original Retina Problem introduced by Kashtan and Alon [KA05], the ANN is responsible for determining either (1) the left *and* right patterns are valid, or (2) the left *or* right patterns are valid. In their version, modularity is helpful when alternating between these tasks because the functions of correctly classifying the left and right patterns are combined. An extension of these tasks, the Retina Left and Right task introduced by Clune et al. [CBM10], removes the need to combine the decision of the modules with a logical operation; instead the ANN must separately judge as valid or invalid each retina’s pattern. Standard HyperNEAT was applied to this task and was shown to struggle with producing the modularity required [CBM10]. In fact, the most difficult of the variant tasks of Kashtan and Alon [KA05] for HyperNEAT in Clune et al. [CBM10] was the Retina Left and Right task, which is thus chosen to explore modularity in this experiment.

As shown in figure 5.2, the substrate is configured with four layers consisting of eight input nodes, eight hidden nodes in layer one, four hidden nodes in layer two, and two output nodes. Their geometric coordinates follow Clune et al. [CBM10]. Note that this substrate has three dimensions (x, y, z) . Each of the inputs are set to either -3.0 or 3.0 , indicating off or on for each retina input. The left and right outputs specify the classification for the left and right retinas, respectively, where values close to -1.0 indicate invalid and values close to 1.0 indicate valid inputs. Each of the 256 possible patterns is presented to the retinas and

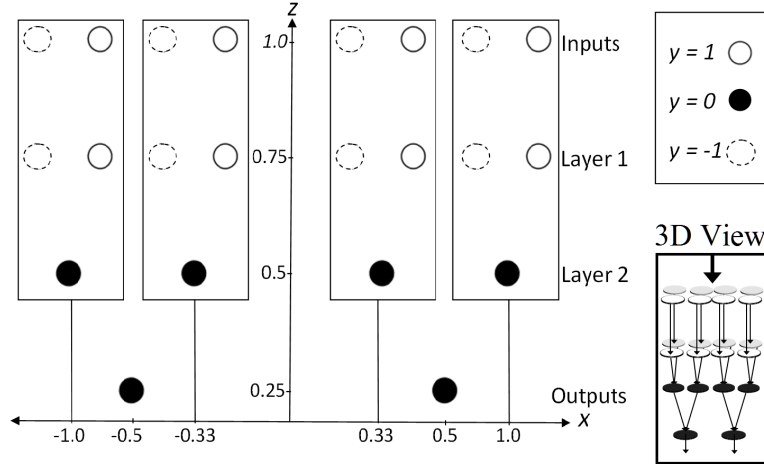


Figure 5.2: Substrate Configuration for the Retina Problem. The substrate configuration is identical to the setup of Clune et al. [CBM10]. The substrate consists of four layers, with the inputs at $z = 1.0$, the first hidden layer at $z = 0.75$, the second hidden layer at $z = 0.5$, and the outputs at $z = 0.25$. Feed-forward connections are allowed between neighboring layers. Nodes in each z layer are placed at an x, y -coordinate. The y coordinates are indicated by the different circle patterns. These patterns are solid-line circles for $y = 1.0$, filled black circles for $y = 0.0$ and dotted line circles for $y = -1.0$. The left and right sides of the retina are reflected through symmetric coordinates on the negative and positive sides the x -axis. This configuration provides a definite division between left and right modules at $x = 0$.

the fitness of the solution is inversely proportional to the summed distance of the outputs from the correct values for all patterns. This setup follows Clune et al. [CBM10].

Eight treatments for controlling thresholding were evaluated on this task. The *uniform threshold* is evaluated with both low (0.3) and high (1.3) threshold settings. The *dynamic distance threshold* is evaluated with a constant of 0.45. Finally, three treatments for *HyperNEAT-LEO* are examined in which evolution is not seeded, evolution is seeded with a global locality pattern, and evolution is seeded with a locality pattern only along the x -axis. The seeded locality pattern is a CPPN with three hidden nodes with Gaussian activation functions that input Δx , Δy , and Δz individually. These hidden nodes are then connected to the output that specifies whether a connection is expressed with a bias of -3.0 .

The activation function of this output is a step function that outputs 1.0 when the input is greater than 0.0 and outputs 0.0 otherwise. In this way, the seed specifies that only connections that are local may be expressed (as in figure 5.1). Slight perturbations of the seed in the initial generation provide a variety of local connectivity patterns.

The alternative seed, which specifies locality along the x -axis only, is similar to the complete locality seed, except that the hidden nodes for Δy and Δz are absent. Finally, this x -locality seed structure is also inserted into both the low and high uniform threshold variants to determine whether a locality bias *alone* (i.e. when it is connected directly to the weight output) is sufficient to induce modularity, as opposed to biasing the separate Link Expression Output with locality.

Finally, the generalizability of locality is examined by applying HyperNEAT-LEO with the global locality seed to the RoboCup Keepaway domain (Chapter 4). RoboCup Keepaway is a challenging and complex machine learning task without any definite bias towards modularity, unlike in the Retina problem. Both standard HyperNEAT and HyperNEAT-LEO with the global locality seed are trained on the standard Keepaway benchmark setup [SSK05], which is three keepers versus two takers on a $20\text{m} \times 20\text{m}$ field in the C# RoboCup simulator described in Appendix A. Both represent the state of the field with the BEV. The parameters (Appendix B) for each approach are identical, differing only in how connectivity is determined. The standard HyperNEAT uniform threshold value is 0.3.

5.1.4 Results

In the Knight Joust domain, the average of 100 runs of 100 generations each found no difference in performance among the different methods. Knight Joust fitness is calculated by the number of forward moves plus a bonus of 20 for reaching the goal. Each of the methods found solutions that reached the goal and achieve an average fitness of 32. However, significant differences were found in the connectivities of the final solutions. In the uniform threshold approach, 86% of the total connections in the ANN are expressed. The dynamic distance threshold and LEO both average 61% of total connections expressed, indicating that they both limit the expression of connections.

Examining the percentage of connections expressed at various distances reveals how the thresholds change the distribution of connections in the geometry. For each of the 100 runs, the final solution is analyzed to determine the number of connections expressed within specific distance ranges by 0.1 intervals, and the aggregate results are averaged together (figures 5.3, 5.4, 5.5). The uniform threshold maintains a high degree of connectivity no matter the length of the connection, ranging from 96% of connections length in the range $[0.0 - 0.1]$ to 82% of connections length 1.9 or greater, demonstrating only a limited tendency to reduce connectivity by distance (figure 5.3). In contrast, the distance threshold maintains a high degree of connectivity for local connections, with 99% of connections expressed in the range $[0.0 - 0.1]$, but significantly curtails longer distances. Only 1% of connections length 1.9 or greater are expressed (figure 5.4). Interestingly, thresholds determined by the LEO lower connectivity for all distances equally, expressing 61% of connections for all distances, which

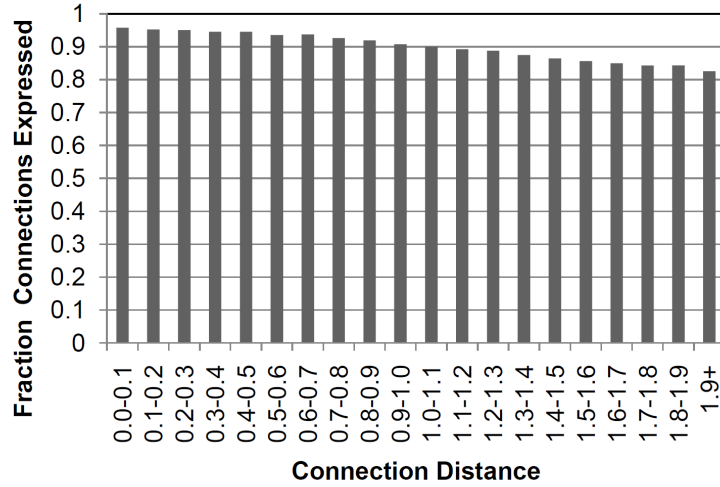


Figure 5.3: Histogram of Connections by Distance for the Uniform Threshold in Knight Joust. The uniform threshold applies to all connections equally regardless of distance. Results are averaged over 100 runs. The uniform threshold allows a high degree of connectivity regardless of distance, never dropping below 82%. However, there is a small change of 14% between the extremes of the distance groups.

suggests no particular pattern by distance (figure 5.5). These results demonstrate that it is possible to constrain the expressed connectivity of the ANN, but for Knight Joust there may be no pressure towards a particular distribution of connections because all were effective.

In the Retina Left and Right problem, the thresholding approaches are evaluated over 100 runs consisting of 5,000 generations of evolution. The performance is recorded for the champion of each generation as the percentage of the 256 patterns the ANN correctly classifies for both the left and right retinas (figure 5.6). As previously shown [CBM10], standard HyperNEAT with the uniform thresholding approaches (unseeded or seeded) fails to achieve high performance: The low threshold converges to 79% ($sd = 4\%$) correct on average and the high threshold reaches 82% ($sd = 8\%$) correct. Interestingly, both the locality-seeded standard-HyperNEAT with a uniform threshold and the dynamic distance

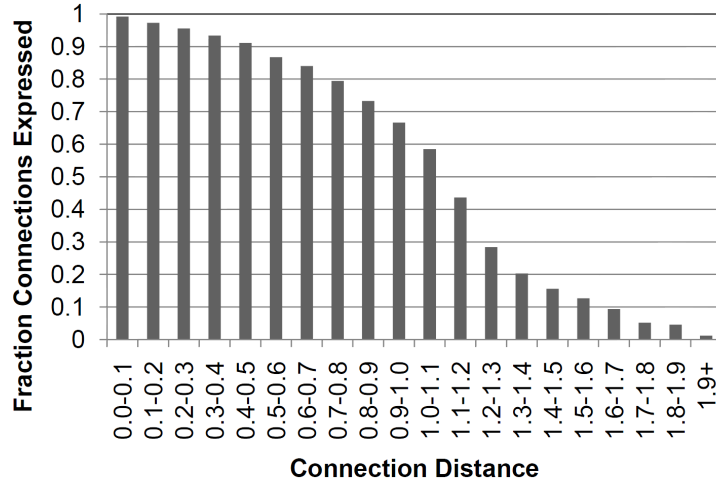


Figure 5.4: Histogram of Connections by Distance for the Dynamic Distance Threshold in Knight Joust. The distance threshold increases in value as distance between nodes increases, decreasing the number of expressed values for longer distances (averaged over 100 runs). The distance threshold allows a high degree of connectivity for close connections, but the percentage of expressed weights decreases to 1% for the longest distance group.

threshold that might be assumed to encourage modularity *decrease* performance, reducing the average performance to 76% ($sd = 2\%$). The HyperNEAT-LEO without a locality seed increases performance to 84% ($sd = 8\%$), but does not significantly outperform the high uniform threshold without a locality seed. Only HyperNEAT-LEO with locality seeds significantly ($p < 0.001$) outperform all other approaches on average, reaching 95% ($sd = 7\%$) and 99% ($sd = 5\%$) accuracy, respectively. Note that an attempt to learn the same task with a direct encoding by Clune et al. [CBM10] yielded only an average of 80% correct answers.

The differing performance of each approach can also be seen in how frequently they solve the problem perfectly (i.e. correctly classify 100% of the patterns), shown in figure 5.7. Of the 100 runs given to them, the uniform thresholds (low and high) solve the problem only 2% and 12% of the time, respectively. The locality-seeded standard-HyperNEAT and the

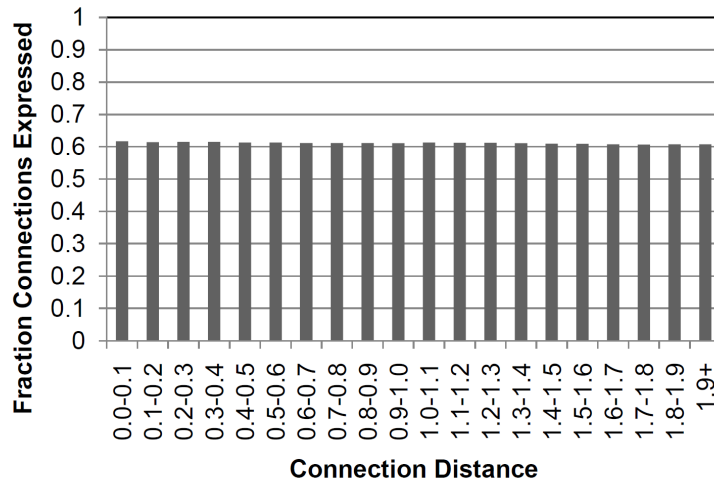


Figure 5.5: Histogram of Connections by Distance for the LEO in Knight Joust. Link expression is generated as a function of geometry by the evolved CPPN with LEO (averaged over 100 runs). On average in Knight Joust, the LEO limits connectivity to 61% for connections of all lengths.

dynamic distance threshold *never* find a perfect solution. Evolving the LEO without a seed solves the problem as often as the high uniform threshold, 12% of the time. The main result is that seeding the LEO with locality significantly improves the ability to find the perfect modular solution. The complete locality seed solves the problem 67% of the time and the seed that specifies locality along the x -axis increases the rate of finding the perfect solution to 91%. These results support the hypothesis that the CPPN itself should be given control of connectivity from a seed that begins evolution with significant locality.

A closer look at the structure of these final solutions gives insight into how and why they succeed or fail. For the uniform thresholds and the non-seeded HyperNEAT-LEO, the common outcome is to prefer fully-connected or near fully-connected patterns (figure 5.8). These patterns indicate that the struggle between finding modularity and establishing the correct pattern for each retina is dominated by optimizing the weight patterns, overriding

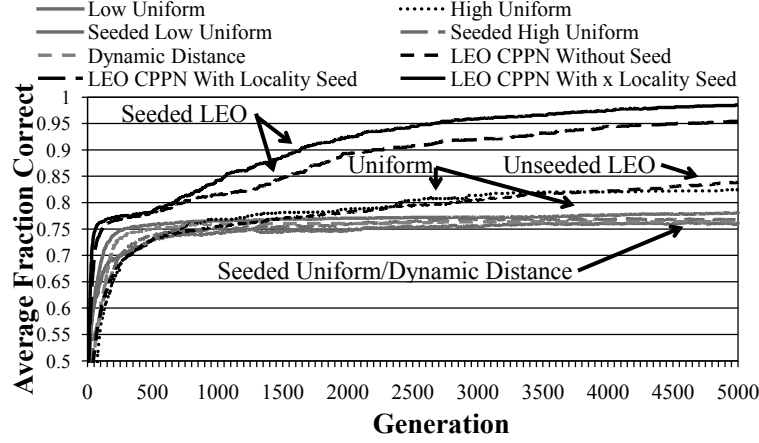


Figure 5.6: Average Percentage of Correct Classifications for Thresholding Methods. The fraction of correct classifications of the 256 patterns is averaged over 100 runs for each of the approaches. Each run lasts 5,000 generations and consists of a population of 500. The uniform and dynamic distance thresholds converge to sub-optimal values. The non-seeded LEO CPPN threshold is still improving at generation 5,000, but is not significantly outperforming the high uniform threshold. However, both the seeded LEOs exceed other approaches' performance with significance $p < 0.001$, demonstrating the importance of locality to the evolution of modularity.

movement towards modularity. Indeed, biasing the *weights* towards locality hinders the search for modularity by providing strong local optima. The distance threshold produces the opposite effect in its patterns (figure 5.9), leading to greater modularity yet less optimal weights. Only when locality is provided through seeded LEO does modularity become the prevailing pattern, while allowing the weights to be optimized separately (figure 5.10). Hence, clearly separated structures emerge consistently. This increased separation can be seen in the average number of left-right side connections in the solution networks (figure 5.11). The low uniform threshold averages 22.74 cross connections in the final solution. The high uniform threshold decreases the amount of cross connections to 13.41. The distance threshold, while underperforming compared to the other approaches, does control connectivity with an average of 13.6 left-right connections. LEO without a bias allows more left-right

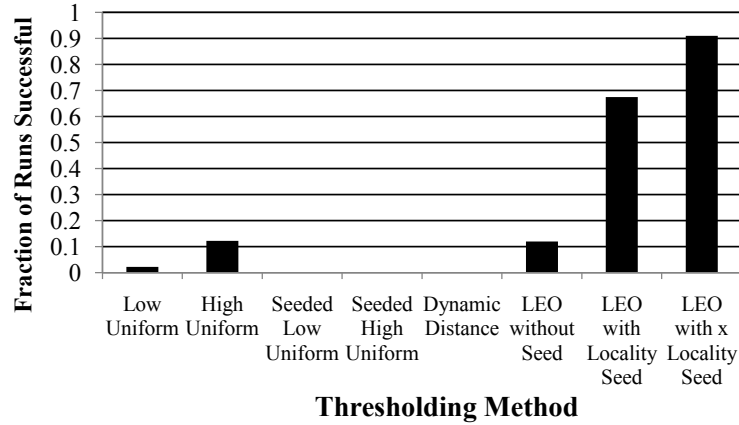


Figure 5.7: Fraction of Runs that are Perfect by Thresholding Method. The fraction of 100 runs that are successful in correctly classifying all of the 256 possible patterns is shown for each of the thresholding approaches after 5,000 generations. The seeded uniform and dynamic distance thresholds find the solution the least often, with success rates of 0%. The low uniform threshold is able to find a solution in 2% of the runs. Seeding non-LEO CPPNs with locality provides no extra benefit while the non-seeded LEO CPPN threshold matches the high uniform threshold; both find the solution in 12% of the runs. The global locality seed improves the odds of finding the solution to 67%, demonstrating how essential the concept of locality is not only to achieving a high fitness, but to finding the exact solution. The x -axis locality seed improves upon the locality seed, finding the solution in 91% of the runs, confirming the intuition that the solution’s best opportunity for modularity in this domain is along the x -axis.

connectivity at an average of 34.95 connections. LEO with global locality seed and LEO with x -locality seed successfully curtail the left-right connectivity with average of 11.25 and 2.9, respectively. More interesting is the frequency of solutions with zero connections between the left and right sides. Validating the inference that separated structures emerge, 64% of runs with the complete locality seed and 85% with the x -locality seed produce solutions with no crossing connections whatsoever between the left and right modules; all the other methods *fail* to produce such structures in more than 75% of runs (figure 5.12).

Finally, both standard HyperNEAT and HyperNEAT-LEO with global locality seed are compared in the RoboCup Keepaway domain. The results are averaged over 100 runs

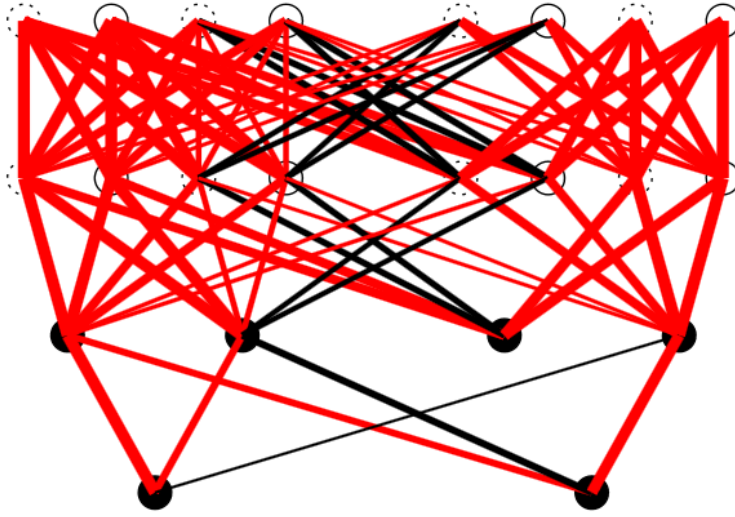


Figure 5.8: Typical Connectivity Pattern of Uniform Threshold. The uniform threshold fails to generate modularity. Instead, it consistently produces near fully-connected networks. These patterns are similar to those produced by the LEO without a seed. Optimizing weight patterns overrides the creation of modular structures. Red (light) lines are negative weights, black (dark) lines are positive weights, and line thickness indicates weight strength.

consisting of 60 generations each. Interestingly, despite lacking a definitive bias towards locality in the RoboCup Keepaway domain, the HyperNEAT-LEO with global locality seed significantly ($p < 0.01$) improves asymptotic performance by 0.89 seconds compared to the standard HyperNEAT. Furthermore, HyperNEAT-LEO learns faster, reaching the asymptotic performance of standard HyperNEAT in seven generations compared to 21 for standard HyperNEAT (figure 5.13). Insight into why performance differs is seen in the expressed connectivity of each approach (figure 5.14). In standard HyperNEAT, connectivity of both the champions and the population is near fully connected at all generations. The population average declines from 157,188 connections to 116,610 in the first few generations, but then increases back to near fully connected. In contrast, HyperNEAT-LEO with global locality seed begins at low average connectivity in both the population and the champions. The

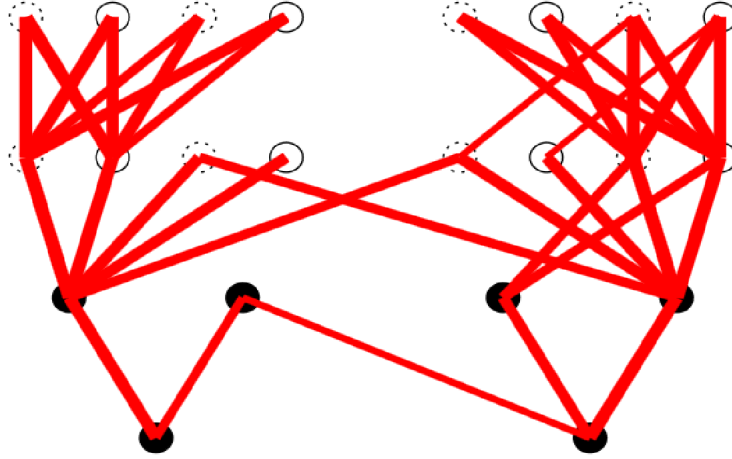


Figure 5.9: Typical Connectivity Pattern of Dynamic Distance Threshold. The dynamic distance threshold generates apparent modularity, but has difficulty finding the correct weight patterns. The problem is that the dynamic distance threshold impacts not only when connections are expressed, but the weight patterns that are created. Because longer distances mean higher thresholds, higher weight outputs become necessary, resulting in a movement towards extreme output values.

connectivity of the champions begins at an average of 7,360 connections and increases to 57,080 over the course of evolution. Similarly, the average population connectivity of the HyperNEAT-LEO begins at a low connectivity of 1,887 connections, increases to 67,402 by generation 11, and finally ends at 66,006 connections. Thus HyperNEAT-LEO enables improved performance by an initial limitation of connectivity that is gradually increased.

5.1.5 Discussion

Modularity is a key component in large and complex structures, in addition to regularity and hierarchy [Lip07]. HyperNEAT is able to generate regular patterns through a hierarchy

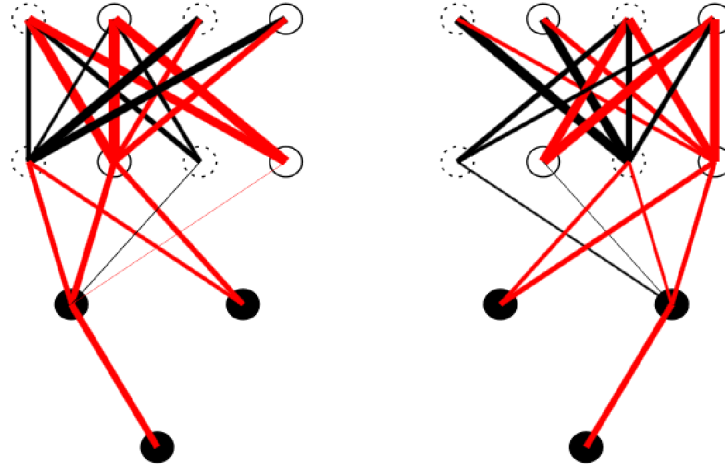


Figure 5.10: Typical Connectivity Pattern of LEO with Locality Seed. Modularity is commonly found when HyperNEAT-LEO is given the concept of locality. Once modularity is found, the regularities needed to solve the task for each module can be discovered in the weight pattern. This separation is possible because HyperNEAT-LEO specifies the pattern of weights and the pattern of connection expression through different outputs of the CPPN. Thus both can be evolved independently, freeing evolution from the burden of searching for a weight pattern that has both the correct weight settings to solve the problem and the necessary modularity.

of function compositions, but has faced a challenge producing modular patterns [CBM10].

By modifying HyperNEAT such that the expression of connectivity is independent of the pattern of the weights, modularity and regularity can be made independent from each other.

Thus both of these key features can be optimized separately. Enhancing this idea, search can be biased by seeding evolution with geometric concepts, such as locality.

One way to understand why determining whether a connection is expressed should be independent from determining its weight is to consider how connection expression and weight determination might interact when they are both determined by the same variable, as in traditional HyperNEAT. In effect, the need to suppress connections in some parts of the network can lead to *distortions* in the pattern of weights, which are themselves a function of

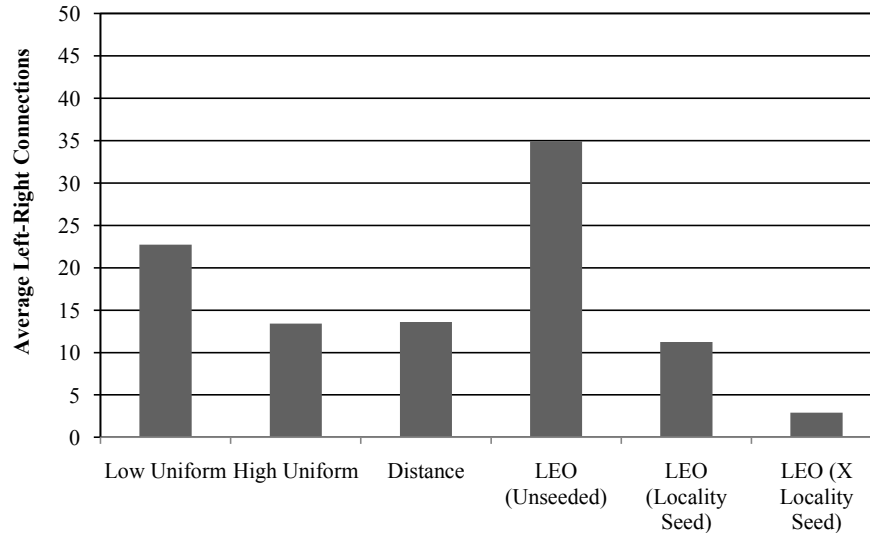


Figure 5.11: Average Number of Left-Right Connections in Retina Solutions. The average count of left-right connections across the final champions of the 100 runs is shown for each of the thresholding approaches after 5,000 generations. The unseeded LEO and low uniform threshold constrain left-right connectivity the least, with averages of 34.95 and 22.74 respectively. The high uniform threshold, dynamic distance threshold, and LEO with global locality seed limit the average left-right connectivity similarly, ranging from 11.25 (LEO with Locality Seed) to 13.6 (dynamic distance threshold). Seeding LEO with x -locality generates the best limitation of left-right connections along the x -axis, averaging only 2.9 connections. Biasing LEO towards locality provides the ability to limit connectivity. The addition of the locality seed significantly decreases the left-right connectivity compared to all other approaches.

the network geometry. In struggling to balance these two demands through a single function, the CPPN ultimately succeeds at neither. This insight suggests that the traditional thresholding technique in HyperNEAT was probably not the best choice in retrospect, although it has been in use for several years [DS07, GS07]. Nevertheless, the good news is that many interesting new structures may evolve now that this problem is uncovered.

An exciting implication of this work is that both connectivity (i.e. which connections are expressed) and connection weights can be indirectly encoded as a *function of geometry* to provide key prerequisites to generating complex structures. The ability to manipulate the

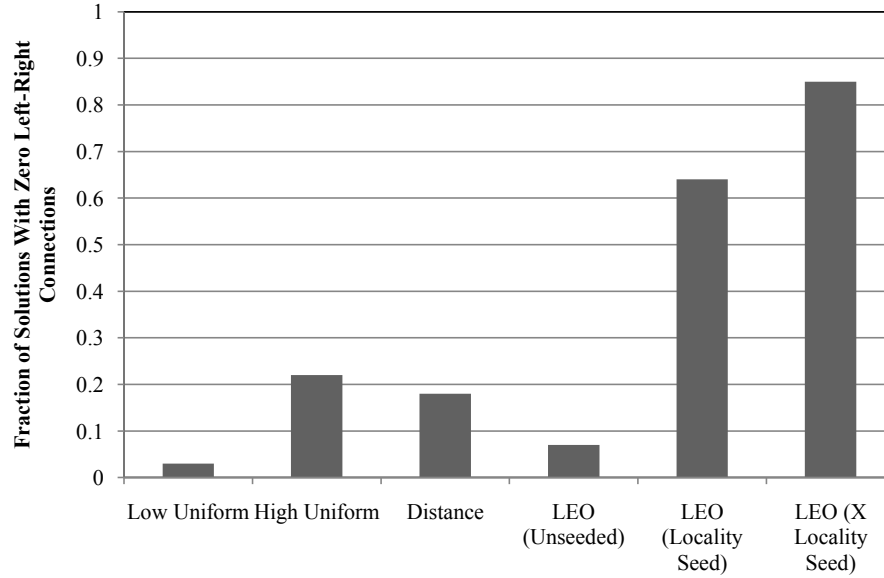


Figure 5.12: Fraction of Retina Solutions that Contain Zero Left-Right Connections. The fraction of final champions of the 100 runs that contain zero left-right connections is shown for each of the thresholding approaches after 5,000 generations. The unseeded LEO and low uniform threshold discover solutions with zero left-right connections the least, at rates of 0.07 and 0.03 respectively. The high uniform threshold and dynamic distance threshold find these types of solutions more often, but still at the low rates of 0.22 and 0.18. Seeding LEO with a locality bias (both global and along the x -axis) significantly improves the rate of finding solutions with zero left-right connections. The global locality seed’s rate is 0.64 and the x -locality seed’s rate is 0.85. Thus separating the connectivity decision into the LEO and biasing it towards locality allows HyperNEAT to generate modular solutions more frequently.

expression of connectivity independently of weights provides the ability to create modules. However, modularity itself follows important geometric principles. One of these, locality, is inherent in the natural universe, that is, components that are grouped into a module are necessarily located near each other because of physical constraints. Simulated structures do not necessarily have such constraints placed upon them, but the principles that arise because of such constraints can be helpful in creating structures that resemble those of nature.

Interestingly, the need for an initial bias towards locality to consistently solve the problem also suggests that the path to encoding locality is inherently deceptive with respect to the

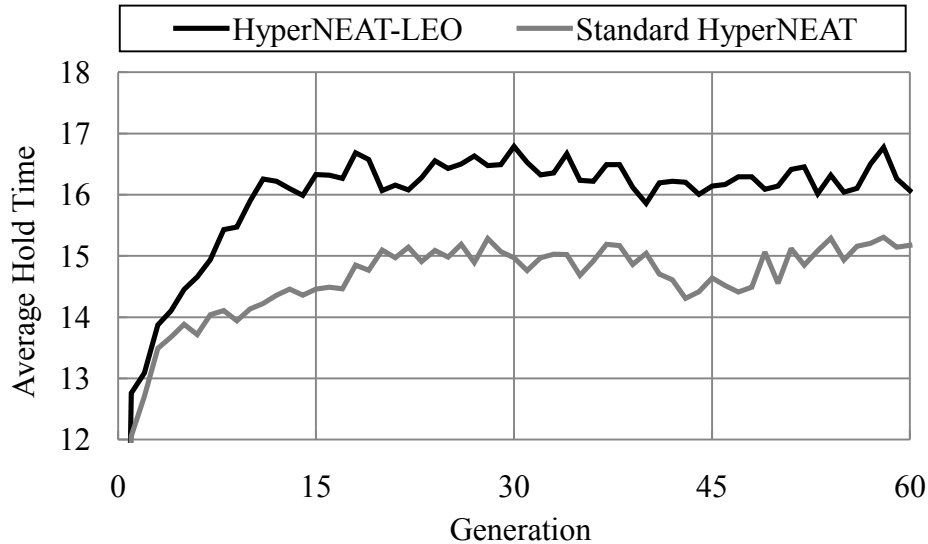


Figure 5.13: Average Champion Keepaway Performance of HyperNEAT with and without LEO. The performance of Keepaway champions for each of 60 generations averaged over 100 runs is shown for both standard HyperNEAT and HyperNEAT-LEO. The seeded LEO hold time of 16.07 significantly ($p < 0.01$) outperforms standard HyperNEAT’s hold time of 15.18 seconds. Additionally, HyperNEAT-LEO learns faster, reaching standard HyperNEAT’s asymptotic performance after seven generations, 14 generations before standard HyperNEAT. Thus the LEO extension with the global locality seed improves learning in the RoboCup Keepaway task.

fitness function in this task. Such deception may turn out common when geometric principles such as locality that are conceptually orthogonal to the main objective are nevertheless essential to achieving it consistently. Thus seeding with the right geometric bias may prove an important tool to avert deception in many domains.

Importantly, it is not enough simply to enforce the concept of locality or bias the structure toward locality to create modules, as demonstrated by the failure of the dynamic distance threshold and the locality-seeded standard HyperNEAT. Instead, evolution should be allowed

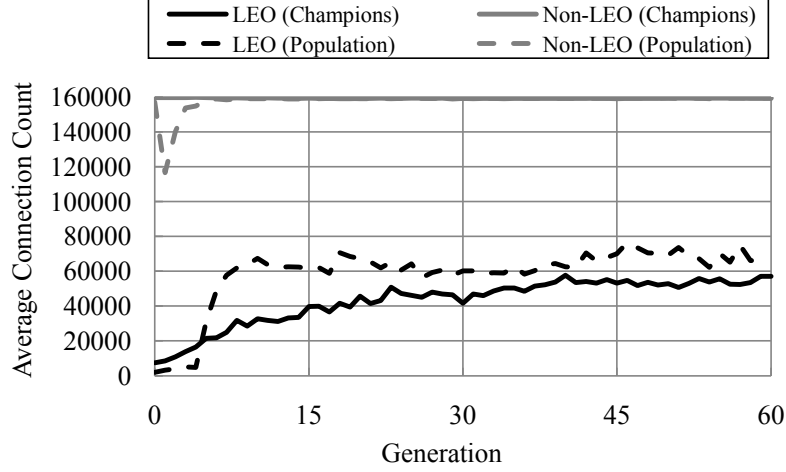


Figure 5.14: Average Number of Expressed Connections of the HyperNEAT BEV in Keep-away with and without LEO. The average connection count for the population and champions for each of 60 generations over 100 runs. The maximum number of connections is 160,000. Standard HyperNEAT’s champions are always nearly fully connected, never dropping below an average of 159,000 connections. The average population connectivity in standard HyperNEAT decreases to 116,610 connections initially, but then returns to near full connectivity and, similarly to the champions, does not drop below 159,000 connections again. In contrast, the global locality-seeded LEO’s average population and champion connectivity begin low at 1,887 and 7,360 connections, respectively. The average connectivity of the champions increases gradually to 57,080, which is significantly ($p < 0.001$) less than standard HyperNEAT. Additionally, the population average of HyperNEAT-LEO jumps to 67,402 connections by generation 11 and then remains stable, ending at 66,006 connections, which is again significantly ($p < 0.001$) less than standard HyperNEAT. Thus separating the connectivity decision and biasing it towards locality allows HyperNEAT to constrain connectivity, begin with a minimal connectivity, and gradually add connections over time.

to find its own separate rules for modularity. This insight gives hope that HyperNEAT can now be applied to new domains that may benefit from modular neural organization.

5.1.6 Conclusion

This section investigated variations on thresholding connectivity in HyperNEAT, introducing the dynamic distance threshold and Link Expression Output. While HyperNEAT has struggled with modularity in the past, decoupling the weight pattern from whether a connection is expressed facilitated the ability to generate modular ANN weight patterns with HyperNEAT-LEO. Instead of failing to find the solution to the modular Retina Problem, HyperNEAT-LEO with a locality seed is almost always able to find the perfect solution, whose modularity is visually apparent in the network geometry. Thus an extension of HyperNEAT now potentially captures two key features of natural systems: regularities and modularity.

Locality provides a starting point for restraining connectivity to achieve modularity. While enforcing locality through an external threshold does restrain connectivity, the results demonstrate that it is detrimental to the search for a correct weight pattern. Ultimately, providing HyperNEAT-LEO with the principle of locality, but allowing evolution to alter the principle, leads to modular structures. Principles from nature such as locality combined with evolutionary algorithms may help us someday to approach the necessary ingredients to match the complexity of nature. Continuing on the theme of alternative substrate configurations, the next section returns to the BEV to investigate its optimal configuration.

5.2 Nonlinearity in the BEV

Because feed-forward artificial neural networks with no hidden layers (i.e. single-layer perceptrons) are linear classifiers, they can only solve linearly separable problems [MP88]. An addition that allows ANNs to solve nonlinear problems is intermediate hidden nodes between the input and output layers. Until now, the BEV, while containing a large number of inputs, outputs, and connections, has contained no hidden nodes and has only been feed-forward. Thus the decision function determined by the BEV is linear. This section explores adding hidden nodes while keeping the BEV feed-forward.

Adding hidden nodes allows the BEV to compute nonlinear separations, but the question is whether the BEV needs such nonlinearity, e.g. in Keepaway. The BEV’s high-dimensional space could reduce the need for nonlinearity (figure 5.15). For example, because the state variables in 3 vs. 2 Keepaway are transformed into a representation on the 20×20 substrate, the traditional 13 inputs are transformed into the 400 BEV inputs. The transformation discretizes the state space by physical location, thereby allowing each location to have an independent decision function from neighboring regions. Because the pattern of weights are encoded by a CPPN that can include hidden nodes, this pattern of weights can vary nonlinearly with BEV location. Thus the BEV is creating a piece-wise linear approximation of a nonlinear function. The same trick is common in temporal difference reinforcement learning approaches that train very large linear function approximators such as the CMAC in domains that would normally be considered nonlinear [Sut96, SSS01].

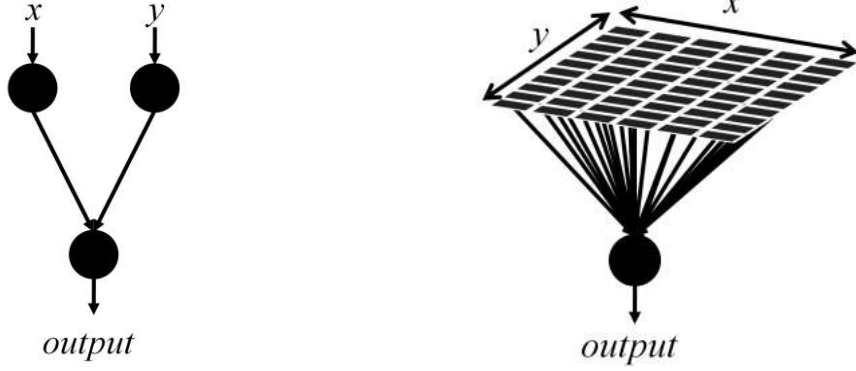
Consider the function $f(x) = wx + b$, where w is a weight and b is a constant bias.

Altering either w or b will only result in a linear function of x . Discretizing the state space of this function f results in a new function $g(\vec{x}) = \vec{w} \cdot \vec{x} + b$. In this new function, \vec{x} is a vector of values (x_1, x_2, \dots, x_n) in which

$$x_i = \begin{cases} 1 & \text{if } x \text{ is in the } i\text{-th discrete component;} \\ 0 & \text{otherwise,} \end{cases}$$

and \vec{w} is a vector of weights (w_1, w_2, \dots, w_n) that associate with the corresponding x_i . Though g is a linear function of \vec{x} , it approximates a nonlinear function of x by varying the weights nonlinearly. As the length of the vector (n) approaches infinite, the discretization becomes more precise and the function g approximates functions of x with increasing accuracy. The challenge for the BEV representation is that multiple x_i may be active at any time. Because g is still a linear function, it cannot compute nonlinear functions of these discrete components themselves. The question is whether the BEV benefits by adding the capability to compute such nonlinear decisions.

To investigate the added benefit of nonlinearity, HyperNEAT-LEO variants with zero, one, and two hidden layers are compared in the RoboCup Keepaway domain. Learning is applied to the standard RoboCup Keepaway benchmark of 3 vs. 2 on the 20m×20m field, as detailed next. In this task, the addition of hidden nodes indeed enhances the BEV’s decision-making capabilities.



(a) Traditional Perceptron

(b) BEV Perceptron

Figure 5.15: Two-Dimensional Single-Layer Perceptron and the BEV Equivalent. A traditional single-layer perceptron with inputs x and y (a) can only solve linearly separable problems [MP88] in the x, y -plane. The BEV (b) transforms the two inputs of the single layer perceptron into a grid of inputs that form an x, y -plane. Though the BEV remains linear, it can now create a piece-wise linear approximation of a nonlinear function across the x, y -plane.

5.2.1 *Experimental Setup*

To evaluate the benefit of nonlinearity, the BEV without hidden nodes is compared with evolving the BEV with hidden nodes. The BEV without hidden nodes computes a linear decision at each output, but the pattern of weights is nonlinear, creating a piecewise linear approximation. The addition of hidden nodes allows each BEV output to generate a nonlinear decision. For example, without hidden nodes, the presence of a player at location (x_1, y_1) and at location (x_2, y_2) could only be linearly combined. Thus a simple exclusive-or (a player is at (x_1, y_1) or (x_2, y_2) , but not both) is not possible to compute. Intermediate hidden nodes allows the BEV to calculate such functions. Both approaches are tested in the

same RoboCup Keepaway domain to evaluate the advantage of adding hidden nodes to the BEV. Recall that in the standard RoboCup Keepaway benchmark three keepers attempt to maintain control of the ball within a $20\text{m} \times 20\text{m}$ region and keep it away from two takers. At each time step, the keeper with the ball must decide whether to hold the ball or pass it to one of the other keepers. The other keepers follow a fixed policy for getting open and the takers follow a fixed policy of pursuing the ball. The goal of learning is to find the best actions for the keeper with the ball to maximize the episode duration (i.e. the hold time).

In the standard BEV configuration for this task (Chapter 4), the BEV consists of an input layer and output layer, each configured into a 20×20 grid. Thus the BEV has 400 inputs, 400 outputs, and 160,000 potential connections. Because this BEV configuration contains no hidden layers, it can only generate linear decisions for each output location in the BEV. This BEV with no hidden layers is compared to two additional BEV configurations, one with one hidden layer and the other with two hidden layers. The hidden layers are configured into grids of size 20×20 identical to the input and output layers. These networks are feed-forward and connections only exist between adjacent layers (e.g. input layer to hidden layer 1). Thus the addition of hidden layers results in 400 to 800 new hidden nodes and an additional 160,000 to 320,000 potential connections. However, the hidden layers allows the BEV to make choices that are not linearly separable, that is, each linear piece-wise approximation of the nonlinear function now becomes nonlinear. The CPPN that represents BEVs that include hidden layers remain the same as previous experiments and the added hidden layers are placed at intermediary z -locations.

HyperNEAT-LEO trains each of these three hidden layer configurations (0, 1, and 2 layers) in the Keepaway task. Evolution is seeded with a CPPN containing three hidden nodes with Gaussian activation functions that input Δx , Δy , and Δz individually and connect to the LEO (Section 5.1). Other parameters are listed in Appendix B.

5.2.2 Results

Results are from the HyperNEAT-LEO BEV trained and evaluated in the standard RoboCup Keepaway benchmark in the C# RoboCup Simulator (Appendix A). Note that (unlike in this section) results reported in Chapter 4 are from HyperNEAT without the LEO. Each generation’s champion is tested for 1,000 episodes and performance is averaged over the champions from 100 runs (figure 5.16). The results demonstrate that BEV hidden layers significantly impact performance in RoboCup Keepaway. The standard BEV approaches an asymptotic performance of 16.07 seconds hold time in the C# simulator. Adding a single layer of hidden nodes increases performance to 17.35 seconds. Two layers of hidden nodes further increases performance to 17.83 seconds. Thus further nonlinearity enhances the BEV’s capability in the RoboCup Keepaway domain. The differences are significant ($p < 0.01$) between no hidden layer and one or two hidden layers, but not significant between one and two hidden layers.

Another consideration in performance is the rate of learning. In adding hidden nodes,

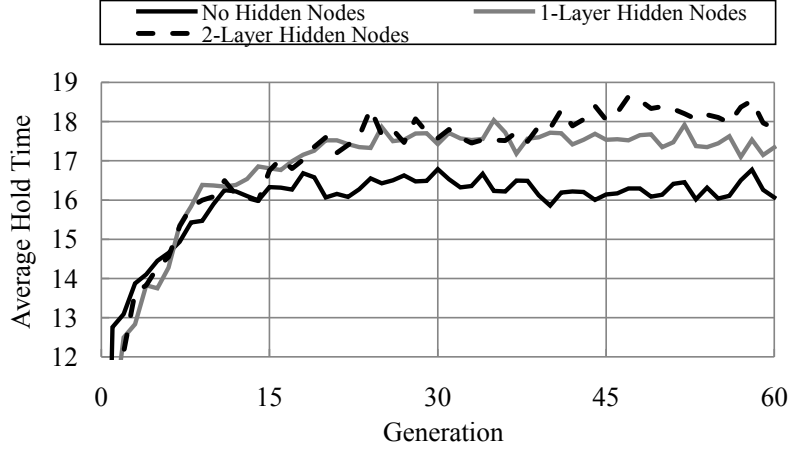


Figure 5.16: Impact of Hidden Layers on Performance in RoboCup Keepaway. Average champion BEV performance of 100 runs at each generation is evaluated in the RoboCup Keepaway task without hidden layers, with one hidden layer, and with two hidden layers. The BEV substrate without hidden layers simply has 400 inputs and 400 outputs. A single hidden layer contains 400 hidden nodes and two hidden layers add 800 hidden nodes. The asymptotic performance of the agents with hidden layers exceeds the performance without hidden layers significantly ($p < 0.01$). The BEV without a hidden layer ends with a performance of 16.07 seconds. A single hidden layer improves this performance to 17.35 seconds and a second hidden layer further increases performance to 17.83 seconds, but no significant difference exists between one and two hidden layers. Interestingly, the increase in complexity and size of the BEV substrate does not negatively impact on the rate at which the task is learned. The BEV substrates with hidden layers do not significantly differ in performance from the BEV without hidden layers, until they exceed the performance without the hidden layer. Thus the addition of hidden nodes positively impacts the ultimate performance of the BEV in RoboCup Keepaway, without negatively impacting the task learning speed.

the complexity and size of the BEV increases (though not necessarily the size of the CPPN that encodes it). A single hidden layer increases the number of nodes by 400 and the potential connections by 160,000. Two hidden layers increases the number of nodes by 800 and the number of potential connections by 320,000. However, despite the increase in size and complexity, there is no significant difference between the fitness of the BEV without hidden layers and those with hidden layers, until the BEVs with hidden layers exceed the

asymptotic performance of the BEV without hidden layers. Thus the learning rate of the BEV is not negatively impacted by the addition of hidden layers.

5.2.3 Discussion

This section investigated adding further nonlinearity to the BEV through hidden layers. While the original approach to the HyperNEAT BEV contained no hidden nodes, it addressed nonlinearity by generating a piece-wise linear approximation of a nonlinear function. Though each output node represents a linear approximation, each location on the BEV is its own linear approximation. Thus the linear approximation varies nonlinearly as the agents move through the state space. However, each output node can still only make a decision based upon a linear combination of its inputs. The results of adding hidden layers demonstrate that the BEV, even though it addresses some nonlinearity through such approximations, benefits from the additional nonlinearity provided by hidden nodes. This result indicates that for complex tasks hidden nodes may be important for the performance of the BEV. A more general insight is that high-dimensional piecewise approximations in some tasks may still require further nonlinearity (e.g. hidden nodes) to find the optimal solution. Interestingly, despite the addition of connections and hidden nodes, the speed of learning the Keepaway task is unchanged compared to the BEV without hidden nodes. This scal-

ability demonstrates the power of the indirect coding and HyperNEAT to effectively train high-dimensional structures, such as a multiple hidden layer BEV.

5.3 Summary

The investigations in this chapter provide a guide to the best way to configure the neural substrate for the BEV. Two key components were investigated: 1) constraining connectivity and 2) hidden nodes. First, results demonstrated that connectivity may be constrained to encourage the creation organized weight patterns. In particular, HyperNEAT with the Link Expression Output (HyperNEAT-LEO) and a locality seed demonstrated the capability to generate ANNs with modular structures. Furthermore, the capability to begin with a minimal connectivity that can be gradually increased promises to enhance the ability of HyperNEAT to learn. More fundamentally, key features of the natural world, such as the principle of locality, can encourage sound designs in artificial structures.

Second, the investigation of nonlinearity in the BEV showed that the addition of hidden nodes can provide an advantage in learning. The initial nonlinear capability, provided by piecewise approximation, benefits from the further nonlinearity through hidden nodes. In the RoboCup Keepaway domain, adding a single layer of hidden nodes results in a significant increase in the performance of the BEV, and an additional layer increases performance further. Thus this result demonstrates that complex tasks addressed by the BEV may also

benefit from hidden nodes in the configuration of the neural substrate. It is important to note that the addition of hidden nodes did not hinder the rate of learning; thus the only cost of adding hidden nodes is the computational cost of the additional structure.

CHAPTER 6

EXPERIMENTS IN INCREASING RESOLUTION

Traditionally, the number and positions of the nodes in the substrate in HyperNEAT is determined a priori and does not change during evolution, being designed to reflect robot architecture [SDG09, DS10], game board structure [GS10a], body morphology [CBO09] or task division [CBM10]. Often, this fixed substrate is sufficient because no new sensors are needed; instead they are completely specified at the beginning of evolution. The number of nodes in the BEV substrate determines the resolution of the two-dimensional BEV plane. This chapter begins by detailing experiments published in the *JMLR* [VS10a] that demonstrate that increasing the resolution of trained BEVs can provide an advantage in performance. The following two sections present two new techniques for increasing resolution and exploiting the variable resolution of the substrate, which are made possible by the indirect encoding of the BEV. The new techniques include dynamically querying an infinite-resolution substrate and incrementally the substrate’s resolution.

6.1 Transfer of Field Size and Substrate Resolution

The experiments in Chapter 4 demonstrated the BEV’s advantage in transferring to more agents and between different domains. A thorough evaluation of transfer recognizes that there is more than one way to alter a task. One such alternate type of transfer in RoboCup Keepaway is changing the field size or BEV precision.

6.1.1 *Experimental Design*

Transfer from a smaller to larger field is evaluated by testing the best policy trained in 3 vs. 2 on varied field sizes in the original RoboCup Soccer Server simulator. Stone et al. [SSS01] previously investigated this kind of transfer on their best Sarsa solution in an easier version of the Keepaway task that does not include noise. They tested a single high-performing individual that was trained on a fixed field size ($15\text{m} \times 15\text{m}$) not only on the trained field size, but also on the other two field sizes.

To investigate transfer to increased BEV *resolution*, the substrate resolution of the champion individuals from five runs from training on three field sizes ($15\text{m} \times 15\text{m}$, $20\text{m} \times 20\text{m}$, and $25\text{m} \times 25\text{m}$) are doubled in each dimension and then tested again on the same field size. For example, a 20×20 BEV becomes 40×40 , which means that each input represents one quarter as much of the space as before. This BEV quadruples the number of inputs and outputs

while increasing the number of connections by a factor of 16 (from 160,000 to 2,560,000 connections). All results are from testing over 1,000 trials and are averaged over five runs with each run consisting of 50 generations of evolution.

6.1.2 Results

Results in transfer to larger fields are from the best policy trained by the HyperNEAT BEV (which, unlike Sarsa, was subject to noise) on the $15\text{m} \times 15\text{m}$ field size that were trained and tested in the original RoboCup Soccer server (figure 6.1). The results are interesting because they show that the representation can cause performance to vary in unexpected ways. For example, even though larger field sizes are easier, Stone et al. [SS01] report that the performance of the best keepers trained by Sarsa *declines* when they are transferred to larger fields. However, even hard-coded policies, such as Random, Always Hold, or Hand Tuned, increase in performance as field size increases, demonstrating the decreased difficulty of the task. Also, in contrast to Sarsa, when transferred to larger fields, the keepers trained with the HyperNEAT BEV improve their performance (as would be expected) from 5.6 seconds to 11.0 seconds and 13.8 seconds, respectively, and outperform the hand-designed policies (figure 6.1). These improvements make sense because the task should become easier when there is more room on the field.

Another consideration in performance is simply that when the field size is $15\text{m} \times 15\text{m}$,

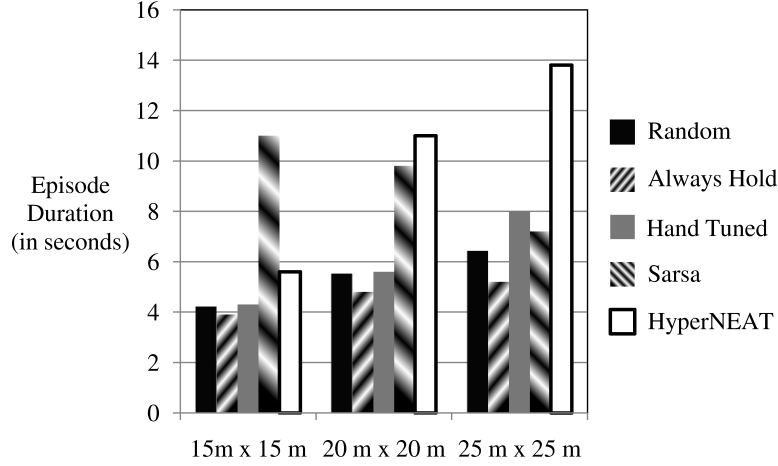


Figure 6.1: 3 vs. 2 Transfer Performance To Larger Field Sizes. Transfer to larger field sizes is evaluated by testing an individual trained on a $15\text{m} \times 15\text{m}$ field on larger field sizes ($20\text{m} \times 20\text{m}$ and $25\text{m} \times 25\text{m}$). The BEV substrate is scaled to maintain the area represented by each discrete unit. Results from [SS01] show that policies trained by Sarsa and transferred to larger fields *decrease* in performance. However, the task is *easier* on larger fields, as shown by fixed policies (Random, Always Hold, and Hand-Tuned) that *increase* in performance as field size increases. In contrast to Sarsa, the BEV learns to outperform fixed policies and transfers to larger field sizes, *significantly improving* performance.

the BEV resolution is *also* at 15×15 , which may be too low to capture the detail necessary to succeed in the task. Confirming this hypothesis, if the BEV is trained at 30×30 resolution on a $15\text{m} \times 15\text{m}$ field, its performance rises significantly, to 7.1 seconds compared to 7.4s for Sarsa when it *is* trained with noise on $15\text{m} \times 15\text{m}$ [SSK05]. This result raises the interesting question of whether resolution can be *raised* above the training resolution without negative impact, as the next experiment addresses.

The result of increasing BEV resolution in Keepaway is that the knowledge learned through the indirect encoding, i.e. the CPPN, is not negatively impacted by later increasing

Table 6.1: Average Performance of the Best Individuals at Different Resolutions. The regularities learned by the indirect encoding are not dependent on the particular substrate resolution and may be extrapolated to higher resolutions. Increasing the number of connections in the substrate by a factor of 16 (by doubling the size of each dimension) does not degrade performance; in fact, it even improves it significantly in some cases.

Training Field Size	Performance	
	Trained Resolution	Increased Resolution
15m×15m	4.6s	5.3s
20m×20m	15.4s	15.9s
25m×25m	16.8s	16.9s

resolution from that at which the BEV was trained. Table 6.1 shows that no matter the field size, even massively increasing the resolution does not degrade performance and can even lead to a *free* performance increase.

For the 15m×15m, 20m×20m, and 25m×25m field sizes, doubling the size of each dimension on average changes performance from 4.6 seconds to 5.3 seconds, 15.4 seconds to 15.9 seconds, and 16.8 seconds to 16.9 seconds respectively. In one instance, on the 20m×20m field, performance improved instantly from 16.6 seconds to 18.9 seconds. The advantage of this capability is that the BEV resolution can be selectively increased while maintaining the same performance, which makes possible further training with a higher resolution BEV.

6.1.3 Discussion

The BEV’s advantage is that the geometric relationships encoded in the CPPN can be extrapolated as the field size increases, thereby extending the knowledge from the smaller field size to the newer areas of the larger field. For Sarsa, such extrapolation is not possible because as field size increases, the new areas represent previously unseen distances for which Sarsa was not trained. Sarsa has no means to extrapolate geometric knowledge from the distances it has seen because, unlike the CPPN, the knowledge learned is not a *function* of the domain geometry (i.e. the geometric relationships on a two-dimensional soccer field). Instead, Sarsa learns a function of the examples presented, which do not explicitly describe the geometry of the domain.

Another important lesson from changing the field size is that BEV performance requires a minimal resolution. When the field size is $15\text{m} \times 15\text{m}$, the BEV performance appears to underperform compared to Sarsa. In part, this difference is because Sarsa was tested originally without noise [SSS01]. A later experiment with Sarsa trained on the $15\text{m} \times 15\text{m}$ with noise [SSK05] shows that its performance is similar to the BEV. However, another factor is simply that when the field size is $15\text{m} \times 15\text{m}$, the BEV resolution is *also* at 15×15 , which may be too low to capture the detail necessary to succeed in the task. Confirming this hypothesis, if the BEV is trained at 30×30 resolution on a $15\text{m} \times 15\text{m}$ field, its performance rises significantly, to 7.1 seconds compared to 7.4s for Sarsa when it *is* trained with noise on $15\text{m} \times 15\text{m}$ [SSK05].

6.2 Virtual Infinite Resolution by Generation Online

This section explores an extension of HyperNEAT called HyperNEAT with Virtual Infinite Resolution by Generation Online (HyperNEAT-VIRGO). Because the weight pattern of the BEV is indirectly encoded by a CPPN, each object on the BEV has a physical coordinate associated with it. Furthermore, because the weights are indirectly encoded as a function of such coordinates, the BEV can be run at effectively infinite resolution. Therefore, interestingly, instead of creating the substrate a priori as normal (figure 6.2a), the nodes can be regenerated *dynamically* at each tick of the clock based on the current coordinates of the objects (figure 6.2b). Thus only nodes that currently represent objects are considered to exist and connections are queried for their weights (by the CPPN) dynamically for those nodes. This process is outlined in algorithm 3.

In this way, the BEV is not confined to a particular resolution and, in fact, represents the exact location of all objects. This effectively infinite resolution thus increases the accuracy of the representation. Furthermore, it addresses the practical consideration that even modest resolutions produce large substrates when the area of the field is large. For example, playing full RoboCup soccer on a 60m×100m field with a 1m² per node resolution would require a substrate with 12,000 nodes and 36,000,000 connections. Yet because the field only contains 23 objects (11 players for each team plus the ball), the part of the BEV that is activated would be sparse and the majority of nodes and connections would not be required at any given moment in the game. Even in the RoboCup Keepaway domain, the number of inputs and outputs that impact the solution at each time step is a small fraction of the total

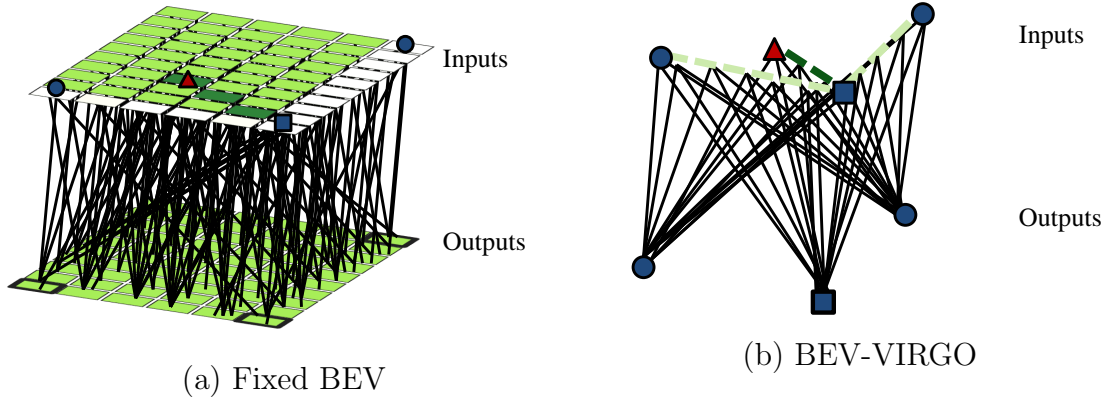


Figure 6.2: Fixed Substrate BEV and the VIRGO equivalent. The original approach to the BEV (a) discretized the field into fixed regions, which are then marked when objects fall within their designated area. The BEV-VIRGO (b) instead generates nodes only at the exact x, y -coordinates for each object and along the relevant paths. The CPPN is then queried for each potential connection between the dynamically generated nodes. Thus the BEV has a virtually infinite resolution (limited by the precision of the data or hardware), by generating the weights online.

(figure 6.3). Querying the substrate dynamically instead creates only the *required* nodes and connections, thereby reducing the computational cost of the BEV substrate.

6.2.1 Experimental Design

To test this infinite resolution, it is compared to fixed-resolution HyperNEAT-LEO in the standard Keepaway benchmark setup [SSK05], which is three keepers versus two takers on a $20\text{m} \times 20\text{m}$ field in the C# RoboCup simulator described in Appendix A. Previously in Section 6.1, increased precision was shown to improve performance in the Keepaway task; thus Keepaway provides an ideal platform for exploring the effect of infinite resolution on

```

1 Initialize population of minimal CPPNs with random weights;
2 while Stopping criteria is not met do
3   foreach CPPN in the population do
4     foreach Time Step of Evaluation do
5       Generate empty ANN substrate  $s$ , set of input coordinates  $i$ , and set of
       output coordinates  $o$ ;
6       foreach Coordinate  $c$  in  $i$  do
7         Create input node at coordinate  $c$  in  $s$ ;
8       end
9       foreach Coordinate  $c$  in  $o$  do
10        Create output node at coordinate  $c$  in  $s$ ;
11      end
12      foreach Possible connection in the substrate do
13        Query the CPPN for weight  $w$  of connection and expression value  $e$ ;
14        if  $e > 0.0$  then
15          Create connection with a weight  $w$ ;
16        end
17      end
18      Run ANN-VIRGO substrate in task domain for current time step;
19    end
20    Ascertain fitness based on performance of ANN-VIRGO substrate;
21  end
22  Produce next generation of CPPNs according to the NEAT method;
23 end
24 Output the champion CPPN.

```

Algorithm 3: HyperNEAT-VIRGO Algorithm

performance. The fixed-resolution BEV remains at 20×20 and the BEV-VIRGO specifies the position of each object to the maximum precision (10^{-1}) of the RoboCup player sensor data. In addition to a comparison of task performance, the computational cost of both approaches is analyzed. Because the BEV-VIRGO is indirectly encoded, both the activation cost of the ANN substrate and the activation cost of querying a connection factor into the total computational cost. Thus the total computational cost is dependent upon both the size of the substrate and the size of the CPPN. A simple measure of this cost is the num-

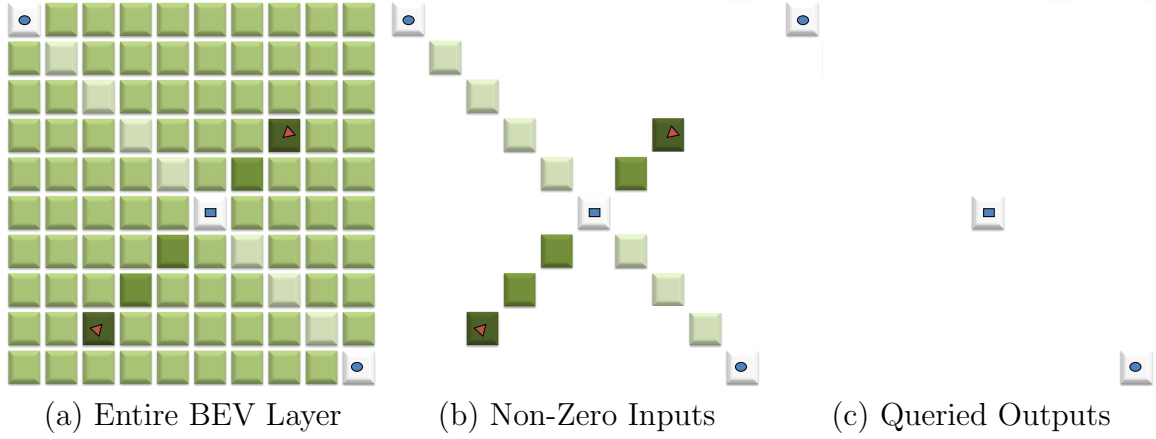


Figure 6.3: Fixed Substrate Keepaway BEV and Nodes that Impact Solution. The Keepaway BEV (a) consists of a grid of nodes that discretize the field into fixed regions. These regions are then marked when objects fall within their designated area. A fraction of the inputs to this fixed BEV substrate (b) have non-zero values. Instead, a majority of the inputs have a value of zero and thus do not impact the output values. Furthermore, a smaller fraction of the output nodes (c) are queried for their values to make a decision, thus rendering the rest of the outputs unnecessary. For example, in 3 vs. 2 Keepaway with a 20×20 BEV, only three out of 400 output nodes are queried on any single time step. Thus activating the full ANN substrate for each time step results in a significant waste of computation.

ber of connections that must be consulted in total, i.e. the number of signals calculated in the CPPN plus the number of signals calculated in the substrate. The sparseness of the RoboCup Keepaway task implies that HyperNEAT-VIRGO should calculate fewer signals per iteration in RoboCup Keepaway than fixed-resolution HyperNEAT.

HyperNEAT-LEO trains each of these approaches in the Keepaway domain. The parameters (Appendix B) for each approach are identical, differing only in how the substrate is generated. Both the fixed BEV substrate and BEV-VIRGO control connectivity through the LEO with the global locality seed (Section 5.1).

6.2.2 Results

Performance results are from the best policies trained by the HyperNEAT-LEO BEV and tested in the standard RoboCup Keepaway benchmark in the C# RoboCup Simulator (Appendix A). Each generation’s champion is tested for 1,000 episodes and their performance is averaged over 100 runs (figure 6.4). The results demonstrate that extending HyperNEAT to dynamically generate an effectively infinite resolution BEV enhances performance. HyperNEAT-VIRGO significantly ($p < 0.01$) improves performance by 1.34 seconds compared to the fixed substrate BEV trained with HyperNEAT-LEO. In addition, the increased precision gained through VIRGO allows the BEV without hidden layers to compete with the fixed substrate BEV with hidden layers. The HyperNEAT-VIRGO BEV (17.41 seconds) performance does not significantly differ from the performance of one hidden layer (17.35 seconds) or two hidden layers (17.83 seconds). Thus HyperNEAT-VIRGO enhances performance and potentially reduces the need for hidden layers.

A key advantage for HyperNEAT-VIRGO is the potential reduction in computation cost. This improvement is measured by counting the number of signals calculated for each time step of the simulation. A *signal* is defined as the multiplication of a single link weight and the output of a node. This value is tracked for each generation’s champion, for the population as a whole, and for the entire run. It is averaged over 100 runs (figures 6.5,6.6). At the start of evolution, HyperNEAT-VIRGO incurs a greater cost: the average number of signals for the population and for champions was 31,016 and 29,625, respectively. In contrast, the fixed substrate BEV’s population starts at an average of 1,887 signals per time



Figure 6.4: Infinite Resolution Performance in RoboCup Keepaway. Average champion BEV performance of 100 runs at each generation is evaluated in the RoboCup Keepaway task at a fixed resolution of 20×20 and the virtual infinite resolution. The effectively infinite resolution BEV-VIRGO performance (17.41 seconds) exceeds the performance of the fixed substrate BEV performance (16.07 seconds) with significance $p < 0.01$. Thus this result confirms the hypothesis that additional precision through increased resolution enhances the performance of the BEV. Interestingly, the BEV-VIRGO with no hidden layer does *not* significantly differ from the performance of the fixed substrate BEV with one and two hidden layers (17.35 and 17.83 seconds, respectively). This lack of difference may imply that the additional resolution improves the BEV’s linear approximation of the nonlinear function.

step and champions begin at 7,360. However, this relationship is reversed by the end of evolution. The average of the final population for VIRGO is 33,318, approximately half the fixed substrate’s 66,006. The champions’ average for VIRGO ends at 34,325, compared to the fixed substrate’s 57,080. Note that VIRGO’s number of signals does not greatly vary between population and champion or beginning of evolution and end. This consistency reflects that the dominating factor in the number of signals calculated for VIRGO is in the

CPPN and not in the substrate, that is, the cost of activating the CPPN for each essential connection on each time step far exceeds the cost of activating the substrate.

Overall, HyperNEAT-VIRGO demonstrates an advantage in the total number of average signals per time step for an entire run (figure 6.6). For each run, the sum total of weights activated per time step across all individuals and all generations is recorded and averaged over the 100 runs. In this measure, HyperNEAT-VIRGO calculates an average total of 193,975,863 signals, which is significantly ($p < 0.001$) lower than the fixed substrate BEV’s average total of 352,008,926 signals. This result demonstrates the significant computational advantage that HyperNEAT-VIRGO provides over the course of evolution in this domain.

6.2.3 Discussion

HyperNEAT’s advantage is that the indirect coding can effectively generate large, complex structures, such as the fixed substrate BEV. The insight in this section is that because it is indirectly encoding the ANN structure, these structures need not be generated a priori. Instead, HyperNEAT-VIRGO dynamically generates weights for the ANN substrate for each time step. This technique allows the BEV to express a virtually infinite resolution by querying nodes at the exact positions they are needed, as opposed to the fixed substrate that represents a fixed area with each node. This infinite resolution increases precision

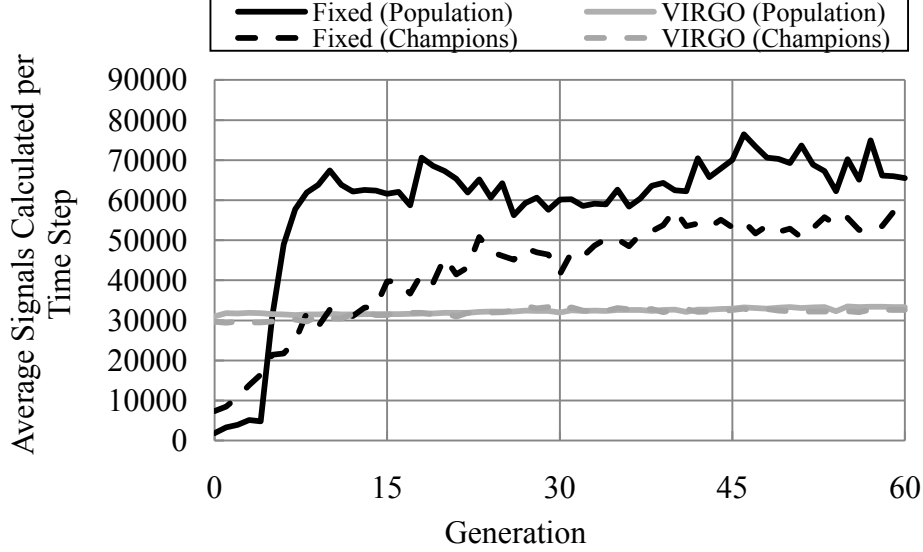


Figure 6.5: Average Number of Signals Calculated for the Fixed Substrate and VIRGO. The average number of signals (i.e. weights activated) per time step of simulation for the population and champion of 100 runs at each generation is tracked in the RoboCup Keepaway task for the fixed resolution of 20×20 and the virtual infinite resolution. The fixed resolution begins with minimally connected ANNs because of the LEO with locality seed and then gradually increases the number of connections in the network. In contrast, VIRGO does not vary largely over the generations or between the population average and champion average. The average number of signals calculated per time step for the population increased from 31,016 to 33,318 and the champion average increased from 29,625 to 34,325. Thus the VIRGO begins with a higher computation cost in signals calculate, but ends with a lower cost compared to the fixed substrate. This trend reflects that the dominating factor in signals calculated per time step for VIRGO is in the CPPN and not the ANN substrate.

and enhances the performance of the BEV in the Keepaway domain. Additionally, because HyperNEAT-VIRGO generates weights on-demand for only the required nodes, there are a significant savings in computational cost. Thus HyperNEAT-VIRGO is a potential path to scaling the BEV to problems of increased complexity or requiring a large fixed substrate. Of course, HyperNEAT-VIRGO capitalizes on the sparse subset of inputs and outputs needed at any given moment. In domains requiring a higher proportion of input and output nodes

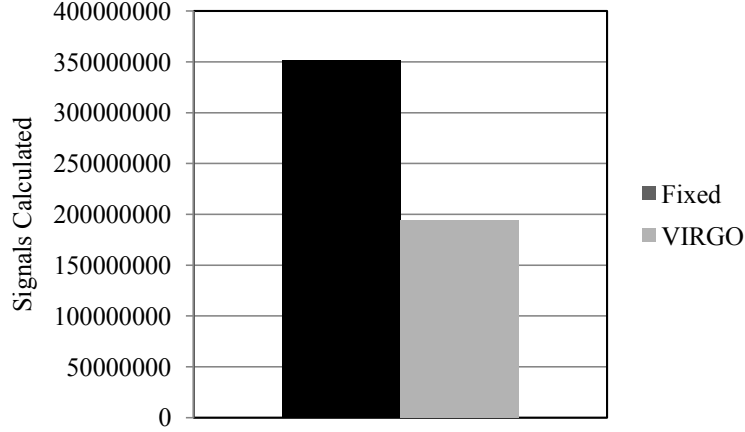


Figure 6.6: Total Number of Signals Calculated for the Fixed Substrate and VIRGO. The average of the sum total signals calculated across all individuals and all generations for 100 runs is shown for the fixed resolution of 20×20 and the virtual infinite resolution in the RoboCup Keepaway domain. The fixed resolution calculates a total of 352,008,926 signals on average for an entire run of 60 generations. In contrast, VIRGO calculates significantly ($p < 0.001$) fewer total number of signals at 193,975,863. Thus the hypothesis that VIRGO provides a computational advantage in signals calculated in the RoboCup Keepaway task over the fixed substrate is confirmed.

its advantage would be less. However, the sparsity in Keepaway is likely representative of many similar domains in which the actions on the field is generally restricted to key locations.

6.3 Phased Continuous Substrate Extrapolation

Previous experiments have shown the effectiveness of the BEV trained at a selected resolution (20×20 grid and VIRGO). Alternatively, instead of selecting a particular resolution a priori, the resolution need not be fixed, as pointed out by Stanley et al. [SDG09]. Gauci

and Stanley [GS10b] later called such resolution change *continuous substrate extrapolation*. Expanding on this capability, this section explores the approach of a *phased continuous substrate extrapolation* that increases the resolution of the BEV *during* evolution to the advantage of ultimate performance. The progressive increase in resolution creates the potential to optimize weight patterns at a coarse grain and then advance to finer granularity. Reducing the complexity of the weight pattern and scaling to higher resolutions could provide a more tractable search space for HyperNEAT. Furthermore, the computational cost of the high-dimensional BEV is offset by scaling from the lower resolutions.

6.3.1 *Experimental Design*

The BEV has already proven effective in the RoboCup Keepaway domain [VS10a], but at a fixed resolution of 1m^2 per node. This resolution creates a large substrate representing a $20\text{m} \times 20\text{m}$ field, resulting in the substrate containing 800 nodes and up to 160,000 connections. Also, HyperNEAT-VIRGO demonstrated that the task could be learned at an effectively infinite resolution to improve performance.

In this experiment, the performance of the fixed resolution and infinite resolution is compared with HyperNEAT-LEO when it begins with a small substrate, 5×5 , that contains 50 nodes and 625 potential connections and then increments every 15 generations through resolutions of 10×10 , 20×20 , and finally infinite resolution through HyperNEAT-VIRGO.

For example, for the first 15 generations the CPPNs generate connectivity patterns for the 5×5 substrate, then at generation 15 the 5×5 substrate is replaced with the 10×10 substrate until generation 30, when it is replaced with the 20×20 substrate. Each of the approaches (fixed substrate, VIRGO, and phased) are evaluated in the standard Keepaway benchmark setup [SSK05], which is three keepers versus two takers on a $20\text{m} \times 20\text{m}$ field in the C# RoboCup simulator described in Appendix A. After training, each of the phased generation’s champions is evaluated on each of the training resolutions and compared to the fixed resolution and to VIRGO.

Additionally, the computational cost of the three approaches is compared. Because the phased continuous substrate extrapolation increases resolution during evolution, the number of signals calculated varies depending upon the current generation. For example, for the first 15 generations the phased approach trains a 5×5 substrate containing 625 potential connection that is orders of magnitude fewer than the 160,000 potential connections in the 20×20 substrate. Thus the phased continuous substrate extrapolation significantly decreases the number of potential connections over the course of a run.

HyperNEAT-LEO trains each of these approaches in the Keepaway domain. The parameters (Appendix B) for each approach are identical, differing only in how the substrate is generated. All approaches control connectivity through the LEO with the global locality seed (Section 5.1).

6.3.2 Results

Performance results are from the best policies trained by the HyperNEAT-LEO BEV and evaluated in the standard RoboCup Keepaway benchmark in the C# RoboCup Simulator (Appendix A). Each generation’s champion is tested for 1,000 episodes and overall performance is averaged over 100 runs. Phased continuous substrate extrapolation results at each of the resolutions demonstrate that learning at a lower resolution provides an advantage to the higher resolutions, but training at higher resolutions does not benefit lower resolutions (figure 6.7). During the first 15 generations of training on the 5×5 resolution, performance at the 5×5 resolution increased from 4.41 seconds to 6.00 seconds. Similarly, the performance at 10×10 , 20×20 , and VIRGO increased performance by 2.89 seconds, 3.71 seconds, and 3.1 seconds respectively. In contrast, from generation 15 to 60 (when the training resolution is increasing), the 5×5 substrate performance decreases from 6.00 seconds to 5.30 seconds. This pattern repeats for each lower resolution upon transition to higher training resolution. Improvements in the training resolution performance correlate strongly to improvements in the higher resolutions. In particular, the correlation coefficient, r , for training resolution to 5×5 is $r = 0.36$, to 10×10 is $r = 0.62$, to 20×20 is $r = 0.78$, and to VIRGO is $r = 0.82$. Also, as resolution increases at each at each 15 generation interval, performance significantly ($p < 0.01$) increases in conjunction with the resolution increase. Thus the results demonstrate that resolution plays a key role in the performance of the BEV and that knowledge gained at lower resolutions can be transferred to higher resolutions.

Phased continuous substrate extrapolation reaches an ultimate performance of 16.67

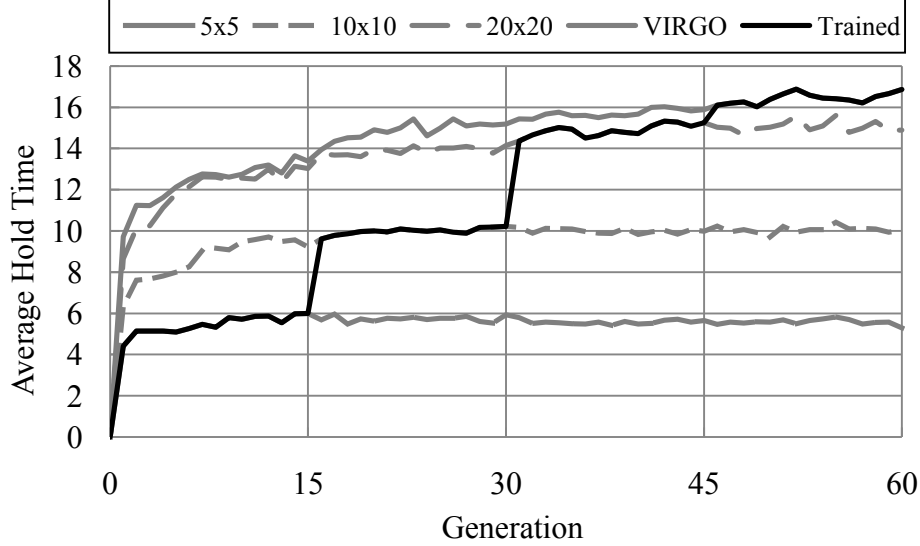


Figure 6.7: Phased Continuous Substrate Extrapolation Performance in RoboCup Keepaway at Each Resolution. The average champion BEV’s performance over 100 runs trained with phased continuous substrate extrapolation is evaluated at each generation in the RoboCup Keepaway task on the fixed resolutions of 5×5 , 10×10 , 20×20 , and VIRGO. The first 15 generations are trained on 5×5 , generations 15 to 30 are trained on 10×10 , generations 30 to 45 are trained on 20×20 , and the final 15 generations are trained on VIRGO. The correlation coefficient of the training resolution to the 5×5 resolution is $r = 0.36$, to 10×10 is $r = 0.62$, to 20×20 is $r = 0.78$, and to VIRGO is $r = 0.82$. Furthermore, each training resolution increase results in a significant ($p < 0.01$) performance increase in the RoboCup Keepaway task. Thus phased continuous substrate extrapolation can effectively learn the Keepaway task by building upon knowledge gained at lower resolutions.

seconds. This result is not a significant improvement over the fixed substrate’s performance of 16.07 seconds and is less than VIRGO’s 17.41 seconds (figure 6.8). Furthermore, the phased increases *inhibit* learning the task. For example, at generation 45, when phased continuous substrate extrapolation ends training at the 20×20 resolution, the performance is 15.24 seconds, but learning solely at 20×20 reaches a performance of 16.14 seconds in the same number of generations. These results indicate that incremental increase in resolution is *not* beneficial to learning the task.

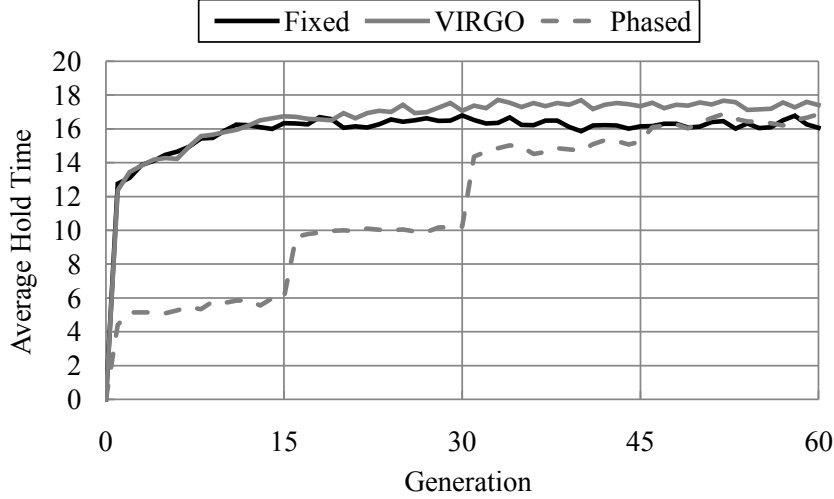


Figure 6.8: Phased Continuous Substrate Extrapolation Performance in RoboCup Keepaway Compared to the Fixed Substrate and VIRGO. The average champion BEV’s performance over 100 runs trained with phased continuous substrate extrapolation is evaluated at each generation in the RoboCup Keepaway task and compared to the performance of the fixed substrate and VIRGO. Phased continuous substrate extrapolation’s final performance at the virtually infinite resolution is 16.67 seconds, which is less than the 17.41 seconds attained by training solely on the infinite resolution. Furthermore, the performance is not significantly greater than training only with the fixed 20×20 resolution substrate to reach the performance of 16.07 seconds. Finally, the phased continuous substrate extrapolation slows the speed of training, taking more generations to achieve the same fitness as when not varying the substrate resolution. Thus phased continuous substrate extrapolation is not suited to enhancing learning in this task.

The connectivity of the phased continuous substrate extrapolation substrates provides insight into the negative impact on performance (figure 6.9). When the substrate is trained at a lower resolution (e.g. 5×5 and 10×10), the substrate is nearly fully connected. However, the connectivity shifts upon switching to the higher resolution 20×20 substrate. At this point, the average number of connections in the champion substrates is 90,739 connections. During training at this resolution, the connection count quickly decreases to an average of 65,541. This results contrasts with the connectivity when fixed substrate evolu-

tion starts minimally and then gradually increases to an average of 57,080 connections. Thus the phased resolution increases deceive the LEO to an increased connectivity that hinders the performance at higher resolutions. It should be noted that the total number of signals calculated over entire runs is significantly ($p < 0.001$) less than the fixed substrate and comparable to VIRGO (figure 6.10) despite the increase in connectivity. The phased approach calculates an average of 187,811,776 signals during a single evolutionary run compared to the fixed substrate's 352,008,926 and VIRGO's 193,975,863. Thus the results confirm that the phased continuous substrate extrapolation can mitigate the computational cost of the high-dimensional substrate.

6.3.3 Discussion

The indirect encoding of HyperNEAT allows the substrate to be altered while the CPPN generating the connectivity remains the same. Thus the substrate can be changed during evolution and the knowledge from previous substrates retained. Phased continuous substrate extrapolation begins with the low resolution substrate that increases at set intervals. The results of experiments in the RoboCup Keepaway domain demonstrated that knowledge gained learning at lower resolutions can transfer to higher resolutions during evolution. Additionally, because lower-dimensional substrates learn in the beginning, the computational cost is significantly less than learning with a fixed high-dimensional substrate. However, results also

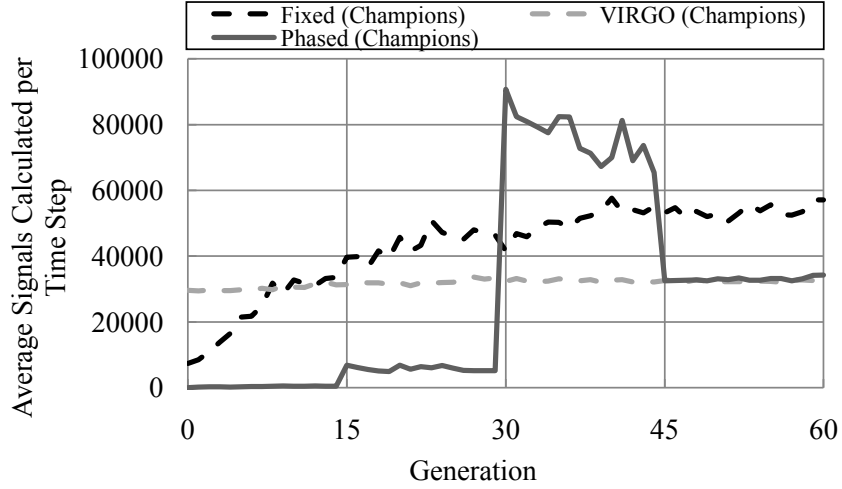


Figure 6.9: Phased Continuous Substrate Extrapolation Champion Signal Counts Compared to Fixed Substrate and VIRGO. Champion BEV’s signals calculated per time step averaged over 100 runs in the RoboCup Keepaway task are shown. The phased continuous substrate extrapolation signals are counted at the trained resolutions (5×5 , 10×10 , 20×20 , and VIRGO). The phased approach results in near fully-connected networks for the first 30 generations at the lower resolutions. The maximum number of connections in the first 15 generations is 525 and the average number of connections increases up to 524. This pattern is different from the fixed 20×20 substrate that begins with a small number of connections and gradually increases to 57,080 out of a possible 160,000. In fact, once the resolution is increased to 20×20 in the phased approach, the number of connections decreases from 90,739 to 65,541. Thus the coarser resolutions cause the LEO to reduce constraints on connectivity that must be increased once resolution increases.

demonstrate that learning is slower and less effective than learning with the fixed resolution or VIRGO substrates. The coarser resolution of the low-dimensional substrates result in an excessive connectivity that must be constrained once resolution is increased. Thus phased continuous substrate extrapolation is not suited towards the RoboCup domain, though it shows some promise for scaling the BEV if the difference in resolutions are smaller than those in this section.

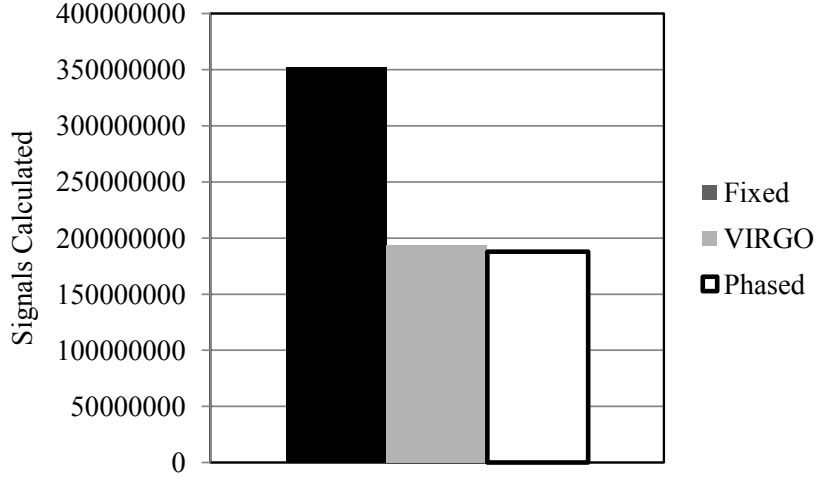


Figure 6.10: Total Number of Signals Calculated for Phased Continuous Substrate Extrapolation, Fixed Substrate, and VIRGO. The average of the sum total of weights activated across all individuals and all generations for 100 runs is shown for the phased approach, fixed resolution of 20×20 , and the virtual infinite resolution in the RoboCup Keepaway domain. The fixed resolution approach results in an average of 352,008,926 signals calculated for an entire run of 60 generations. In contrast, VIRGO calculates a total of 193,975,863 signals. Similarly, the phased resolutions calculate a total of 187,811,776 signals on average. Thus the phased continuous substrate extrapolation provides a significant computational savings compared to the fixed substrate approach, but is comparable to the computational cost of VIRGO.

6.4 Summary

The investigations in this chapter demonstrate key features of resolution in the BEV. First, the BEV was transferred to increasing field sizes in the Keepaway domain. In the Keepaway domain, increasing the field size makes the task easier because the keepers have an increased area to maintain possession away from the takers. As would be expected, performance for the same CPPNs improves as field size is increased. This contrasts with results for Sarsa that show a *decrease* in performance at increased field sizes. As an indirect encod-

ing that learns from geometry, the CPPN allows substrates to extrapolate to increased field sizes. Thus HyperNEAT provides an advantage when transferring to domains that increase in region size, such as increasing the field size in RoboCup Keepaway.

Second, HyperNEAT with Virtual Resolution by Generation Online (VIRGO) shows that a substrate with an effectively infinite resolution can significantly improve performance. HyperNEAT-VIRGO dynamically generates connections for nodes of the BEV at each time step for the precise position of objects. Thus accuracy is maximized and performance is improved. Furthermore, because HyperNEAT-VIRGO creates connections only for the required nodes at each time step, computational cost in the number of signals calculated is reduced. HyperNEAT-VIRGO thus is a potential avenue for scaling to complex problems that require high-dimensional substrates.

Third, phased continuous substrate extrapolation is a technique for gradually increasing resolution to train a high-dimensional substrate. Results indicate that training at lower resolutions can improve performance at higher resolutions. Additionally, beginning at lower resolution and scaling to higher resolutions mitigates the cost of the high-dimensional substrates. However, incrementally increasing resolution adversely impacts performance. The performance at higher resolutions is less than the performance of learning with a fixed substrate or VIRGO and takes longer to reach such performance levels. Thus phased continuous substrate extrapolation has less potential for scaling in the RoboCup domain, but further research may prove fruitful if performance can be improved.

Overall, HyperNEAT has a unique ability to scale solution sizes. In the BEV, this ability

means that the same solution can be applied to larger field sizes or increase precision to improve performance. HyperNEAT-VIRGO extends this ability to dynamically generate an effectively infinite resolution substrate. Thus precision can be maximized and the field size altered online.

CHAPTER 7

SCALING COMPLEXITY: HALF-FIELD OFFENSE

The RoboCup simulated soccer Keepaway domain [SSS01] is well-suited to task transfer experiments because it is a popular RL performance benchmark that can be scaled to different numbers of agents to create new versions of the same task (Chapter 4). Additionally, cross-domain transfer from Knight Joust to Keepaway showed promise (Section 4.2.2). However, both of these domains focus on a single task. In Knight Joust the player is focused on avoiding the opponent and in Keepaway the task is to simply to keep the ball away from the opponents. Full RoboCup soccer contains multiple sub-tasks as well as the overarching task of winning. Players must balance time management, defense, offense, positioning, and scoring. To evaluate the BEV’s ability to scale to complex tasks, it should be tested in a domain where it similarly needs to balance multiple goals.

7.1 Half-Field Offense Domain

A natural stepping stone between RoboCup Keepaway and full RoboCup soccer is the RoboCup Half-field Offense domain [KLS07a, KM11]. In this task, the soccer domain is reduced to a half-field and fewer players, though both larger than in Keepaway. Players are

divided between offense and defense. The offensive team moves the ball toward the goal, maintains possession, and scores goals. The defensive team intercepts the ball and defends the goal. The domain is episodic and each episode ends when the offense scores a goal, the defense intercepts the ball, or the ball travels out of bounds. That way, Half-field Offense expands the complexity of Keepaway by adding the tasks of goal scoring and moving the ball in a particular direction. Note that the goal of Keepaway, keeping the ball away from the takers, is a subtask of the Half-field offense domain. The BEV is well-suited to scale task complexity in this way through task transfer because the BEV can represent the additional agents and the goal without modification.

In the conventional Half-field Offense task [KLS07a], the defense consists of n defenders and the offense has m attackers, where $n \geq m$. One of the n defenders is designated as the goalie and is tasked with defending the goal. Episodes begin with the attackers in a $10\text{m} \times 10\text{m}$ square wherein one side is adjacent to the half-field line and the ball positioned near a random attacker. Two defenders are placed in the center of the square formed by the offense and the rest of the defenders are placed within the goal box. Similarly to Keepaway, only the policy of the attacker with the ball is learned, while the defenders and other attackers follow a fixed policy. The attacker with the ball is given the option of dribbling the ball towards the goal, passing to one of the $m - 1$ teammates, or shooting on goal.

In the 4 vs. 5 Half-field Offense task, if no attacker has possession of the ball, then the closest attacker attempts to obtain possession of the ball. Attackers not the closest to the ball, or without possession of the ball, maintain a trapezoidal formation, which has one side

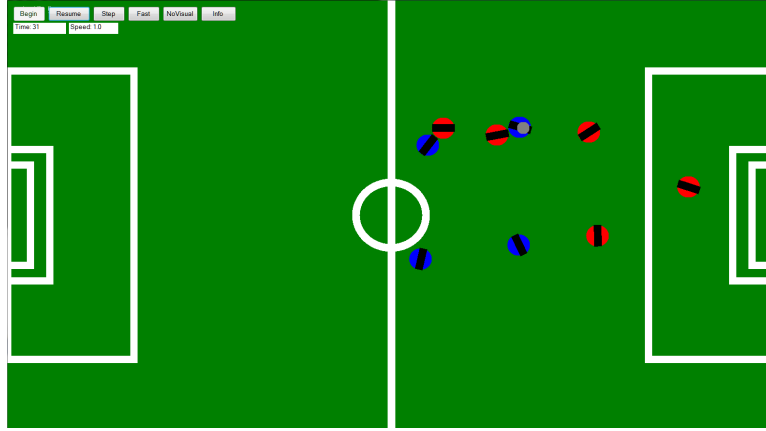


Figure 7.1: Half-field Offense Domain Field. The four attackers (represented by darker circles) begin on the left side of the half-field. The five defense (represented by lighter circles) begin with two defenders in the square defined by the offense and the remaining defenders are placed within the penalty box on the right side, including the goalie. The penalty box is defined by the largest rectangular boundary on the right side, the goal box by the next inner boundary, and finally the goal mouth by the inner-most boundary. The offense must keep the ball in play, that is, within the area defined by the midfield and the top, bottom, and right sides. In addition, the offense must keep the defenders from possessing the ball. The ultimate goal for the offense is to successfully complete shots that pass over the right-hand side of the boundary of the goal mouth.

of length 16m, one side of length 12m, and a width of 10m. For the defensive fixed policy, the defender designated the goalie remains in the goal box and maintains a position between the ball and the goal line. The closest defender to the ball attempts to intercept it and take it away from the offense. The second closest defender to the ball blocks passing lanes to other attacks. Finally, the last two defenders block shooting lanes between themselves and the two attackers closest to the goal (figure 7.1).

7.2 Experimental Setup

To evaluate transfer to Half-field Offense, Keepaway policies trained for 100 generations in the 3 vs. 2 task are transferred to 4 vs. 5 Half-field Offense. The final champion of each of 100 runs of 3 vs. 2 Keepaway seeds evolution for a single 50 generation run of 4 vs. 5 Half-field Offense. Thus the transferred policies are then further trained in the Half-field Offense domain. The performance of evolved policies from seeded runs are compared to policies evolved without seeding. In this experiment, each individual is evaluated for 100 episodes during training and each episode lasts 30.0 seconds. Episodes ending with a goal adds 1.0 to fitness, ending with the goalie catching the ball add -0.3 , ending with other defense taking the ball add -0.2 , ending with the ball going out of bounds add -0.1 , and the episode running out of time adds 0.1 to fitness. This reward scheme penalizes the offense for deciding to shoot foolishly and thus allowing the goalie to control the ball, but also encourages the players to learn ball handling skills by rewarding maintaining possession for the full duration of an episode. This transfer problem is different from previous BEV transfer results (Sections 4.2,6.1) because the *goal* of the domain is now more complicated, as opposed to just the state or semantics of the domain. Thus this experiment demonstrates the BEV’s ability to learn subtasks and then apply the knowledge gained to a full domain. The experiment is run in the C# RoboCup simulator (Appendix A).

As described in Chapter 3 and 4, the HyperNEAT BEV transforms the traditional state representation explicitly to capture the geometry. Each attacker’s position is marked on the input layer with a positive value of 1.0 and defenders are similarly denoted by -1.0 . Paths



Figure 7.2: Visualizing the BEV Input Layer in Half-field Offense. The BEV input layer is marked with the positions of the offense, defense, goal and paths. The offensive player with the ball is the small square, other offense are circles, the takers are triangles, and the diamond is the goal. Positive values are denoted by lighter shades (for offense, the goal, and paths to offense or goal) and negative values are denoted by darker shades (for defense and paths to defense). The middle shade represents an input of 0.0, the lightest is +1.0, and the darkest is -1.0 . The BEV represents the distances and angles in a geometric configuration, allowing geometric relationships to be exploited by HyperNEAT. Paths represent ball position by converging on that offensive player.

are literally drawn from the attacker with the ball to the other players (as in figure 4.1). In addition, paths are also depicted from the attacker with the ball to each corner of the goal with positive values and the goal itself is marked with a positive value of 1.0 (figure 7.2).

In this task, the substrate configurations and resolution (discussed in Chapters 5 and 6) increase in importance. Because the task’s field size is significantly larger (half a soccer field), fine-resolution fixed-substrates become computationally prohibitive. Applying the VIRGO extension (Chapter 6) alleviates computational cost without sacrificing performance.

Additionally, the geometry of the field has greater impact on the actions of the players. In Keepaway, there is little difference among locations in the field because the dominating factor is relative position to opponents and not global positioning. However, positioning in Half-field Offense has meaning because of the goal. Decisions made near the goal could be functionally different from those made near mid-field. Thus modularity could play a role in separating these distinct regions and therefore the LEO extension is included.

HyperNEAT-VIRGO+LEO trains policies in the Keepaway domain and the Half-field Offense domain. The parameters for this experiment are listed in Appendix B. The LEO is initially given the global locality seed (Section 5.1).

7.3 Results

Transfer is evaluated by first training in the standard Keepaway 3 vs. 2 benchmark (Chapter 4) and then seeding the 4 vs. 5 Half-field Offense training with the Keepaway champions. The results with transfer are compared to training without transfer. After training, the champion of each Half-field Offense generation is tested over 500 trials. Performance results are averaged over 100 runs with each consisting of 50 generations of evolution.

Performance results are from the best policies trained by the HyperNEAT-VIRGO+LEO BEV and tested in the standard RoboCup Keepaway benchmark in the C# RoboCup Simulator (Appendix A). The results demonstrate that scaling to the complex Half-field Offense task from the Keepaway task through task transfer improves performance (figure 7.3). Ini-

tial random policies achieve an average 0.016 reward and the transferred policies without further training average -0.010 reward. Thus policies transferred from Keepaway have a negative jumpstart performance. However, upon further training, transfer from Keepaway to Half-field Offense achieves a performance of 0.081 compared to a performance of 0.062 without transfer, significantly ($p < 0.05$) improving asymptotic performance. A performance of 0.081 indicates an improved scoring ability with increased successful shots on goal and a decreased percentage of shots blocked by the goalie. These improved players capitalize on knowledge from Keepaway by effectively maneuvering the ball around the defense while still moving the ball towards the goal. This increased ball handling forces the defense out of position, which is a common strategy in soccer. In contrast, players without transfer do not attempt to move the ball around to force defensive mistakes. In addition, the average total reward received by all generation champions throughout each run after transfer (which is a standard measure of transfer performance; Section 2.5; Taylor and Stone [TS09]) is 3.112, which is significantly ($p < 0.01$) greater than the total reward without transfer of 2.110. The transfer ratio (Taylor and Stone [TS09]) between these approaches total reward is $r = 0.475$. Thus HyperNEAT BEV effectively transfers knowledge that bootstraps performance in the Half-field Offense domain.

The differences between transfer and non-transfer performance can be seen in the particular outcomes from different episodes (figures 7.4, 7.5). The initial champions of runs without transfer end each episode with a goal 22% of the time, the goalie catching the ball 56% of the time, the defense intercepting the ball 12% of the time, and the ball going out of

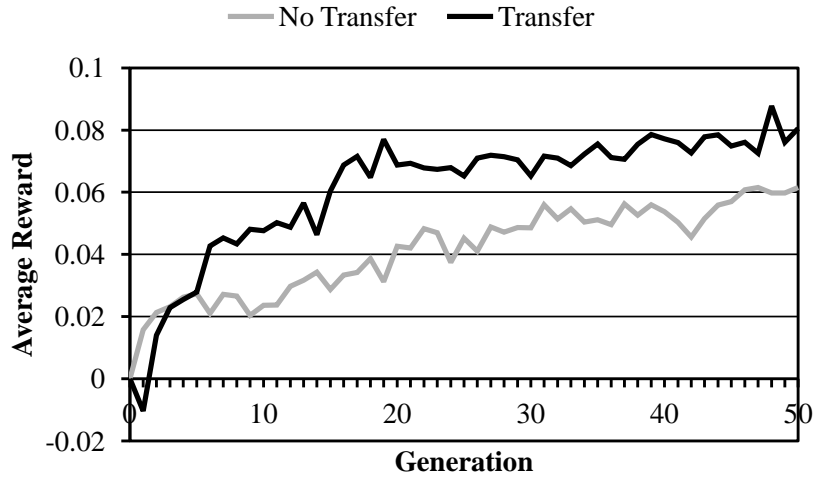


Figure 7.3: Task Transfer Performance in RoboCup Half-field Offense. Average champion performance of 100 runs at each generation is evaluated in the RoboCup Half-field Offense task, with and without transfer from Keepaway. The performance of initial random policies (0.016) exceeds the initial performance of transferred policies (-0.010). Thus transfer from Keepaway to Half-field Offense has a negative jumpstart performance. In contrast, transfer improves asymptotic performance from 0.062 to 0.081 with significance $p < 0.05$. It should also be noted that transfer’s initial negative impact on performance disappears by generation three. Finally, the total reward achieve through transfer (3.112) significantly outperforms total reward without transfer (2.110). These total reward values produce a transfer ratio of $r = 0.475$. Thus transfer from Keepaway to Half-field Offense improves overall performance.

bounds 10% of the time (figure 7.4). The policies transferred from Keepaway initially score goals at a rate of 19%, are blocked by the goalie in 52% of the episodes, are intercepted the defense at a 16% rate, and send the ball out of bounds 13% of the time. The policies transferred from Keepaway are more inclined towards keeping the ball in play, thus resulting in more instances of the defense intercepting the ball or the ball going out of bounds, rather than choosing to shoot on goal and be blocked by the goalie. In contrast, the transferred policies with further training exceed the goal-scoring performance of the non-transfer poli-

cies, have fewer shots blocked by the goalie, and keep the ball in bounds more often (figure 7.5). The final transfer champions score a goal in 27% of episodes, are blocked by the goalie in 51% of episodes, are intercepted the defense in 12%, and allow the ball to travel out of bounds in 10%. Champions from training without transfer score fewer goals (25%), are blocked more (53%), are intercepted less (11%), and allow the ball out of bounds more often (11.0%). Thus transfer enhances the ultimate scoring ability of policies in the Half-field Offense domain.

7.4 Half-field Offense Discussion

Previous results with the BEV representation (Chapter 4) demonstrated that task transfer was facilitated between tasks of similar complexity. The results in this chapter show that the BEV extends to facilitating transfer to tasks of higher complexity. The popular RoboCup Keepaway benchmark has a single learning goal, i.e. keeping possession of the ball. Note that Keepaway significantly differs from Half-field Offense and the Half-field Offense task introduces new complexity, such as directionality on the field. However, Keepaway is a subtask of the RoboCup Half-field Offense domain, that is, maintaining possession of the ball and away from defenders is important in the Half-field domain. Indeed, despite the differences between tasks, the results demonstrate that transferring from Keepaway does improve the ultimate performance in the Half-field domain by a significant amount.

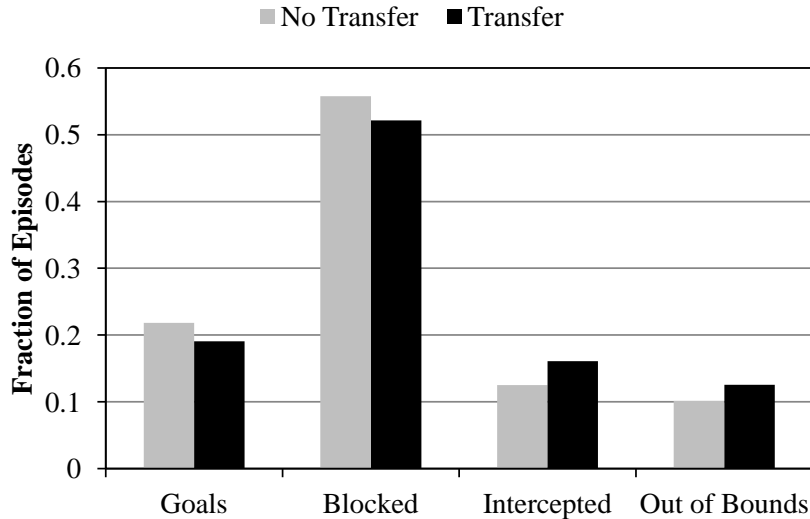


Figure 7.4: Initial Performance of Transferred Keepaway Policies and Random Policies in Half-field Offense. The policies trained in Keepaway and then transferred to Half-field offense episodes result in a goal in 19% of episodes, being blocked by the goalie in 52%, being intercepted the defense in 16%, and the ball traveling out of bounds in 13%, without additional training in the task. In contrast, champions of the initial random populations (without transfer) end episodes with a goal in 22% of episodes, the goalie catching the ball in 56%, the defense intercepting the ball in 12%, and the ball travelling out of bounds 10% of the time. Half-field Offense policies transferred from Keepaway shoot on goal less often than initial random policies. Interestingly, transferred policies have a better capability of keeping the ball in play, indicating knowledge gained from keeping the ball in play in Keepaway bootstraps into Half-field Offense.

The Half-field Offense domain also considers sub-tasks other than strictly keeping the ball away from the opponents. Agents in the domain must also move the ball toward the goal, position for shots on goal, and discover when optimally to shoot on goal, increasing the complexity required by the solution. Interestingly, transferred policies fare differently from initial random policies in different sub-tasks. For example, transferred policies are initially better at keeping the ball in play, but do not shoot on the goal as often as non-transfer policies. Such differing capabilities indicate that training in Keepaway biases the skill set

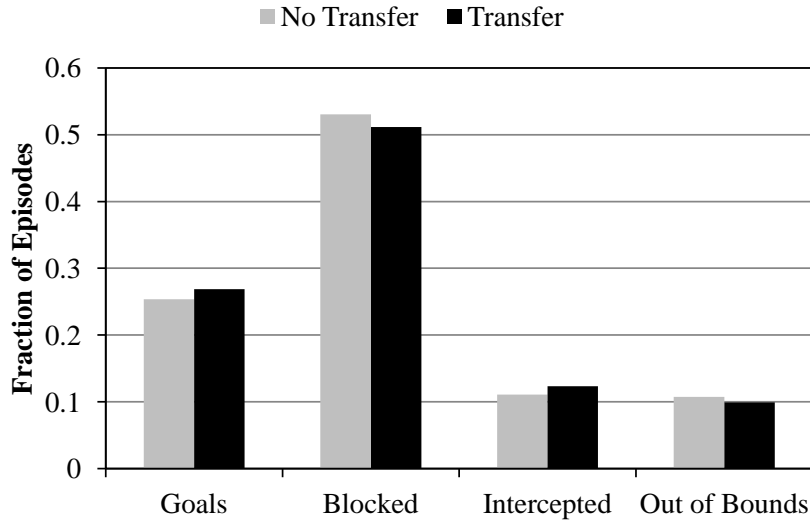


Figure 7.5: Trained Performance of Transferred Keepaway Policies and Trained Policies without Transfer in Half-field Offense. The policies trained in Keepaway, transferred to Half-field offense, and then further trained end with a goal in 27% of episodes, being blocked by the goalie in 51%, being intercepted the defense in 12%, and the ball traveling out of bounds in 10%. In contrast, champions from training without transfer end episodes with a goal 25% of the time, the goalie catching the ball 53%, the defense intercepting the ball 11%, and the ball going out of bounds 11.0% of the time. Thus transfer enhances the ultimate scoring ability of policies in the Half-field Offense domain. In particular, transfer improves the ability to make successful shots, keep the ball away from the goalie, and keep the ball in play.

in a particular direction for Half-field Offense. Because the skills imparted are a subset of the skills required in Half-field Offense, the BEV positively transfers policies from Keepaway. Ultimately these differences are expressed in the behavior of final policies. Policies trained without transfer focus on a shoot first policy, attempting to move the ball forward and shooting as quickly as possible. In contrast, policies with transfer move the ball more to keep the defense away and maintain possession for a longer period of time before attempting to score.

Prior experiments in Half-field Offense have focused on improving multi-agent learning [KLS07a] and properties of the domain [KM11]. Note that direct comparison to these previous results is not possible due to variations in experimental setup and reward schemes, but such prior results do provide context about the domain and approximate good performance levels. Importantly, these prior experiments were applied in a different simulator and were not investigated for their ability to transfer.

Kalyanakrishnan et al. [KLS07a] introduced Half-field Offense and investigated communication between the agents to facilitate multi-agent learning with the Sarsa algorithm. Communication between agents allows learning updates to be shared across all agents, rather than confined to an individual agent. Their experiments in the 4 vs. 5 Half-field Offense task in the original RoboCup simulator improved the success rate (i.e. goals scored) from 23% to 32% by sharing updates and creating an effectively homogenous policy.

Kohl and Miikkulainen [KM11] explored the properties of a 5 vs. 5 Half-field Offense variant; in particular, the idea of *fractured domains* was investigated. A fractured domain is one in which the optimal solution shifts suddenly, that is, the optimal solution is not smooth and possibly discontinuous. Kohl and Miikkulainen showed that an approach shown to address fractured domains (Cascade-NEAT), at a 35% success rate, outperforms approaches that include Sarsa, other NEAT variants, a vanilla genetic algorithm, and the ESP neuroevolution algorithm. A posited explanation is that Half-field Offense and related domains are indeed fractured. Interestingly, the BEV is able to successfully learn in this domain and thus the

BEV may address fracturing in domains through its representation of state.

7.5 Summary

In summary, the BEV simplifies task transfer by representing state in a two-dimensional plane. As task complexity increases, whether the objects increase in number or new objects are added, they can be drawn on the existing BEV substrate. In contrast, traditional representations must expand the state space with the addition of new objects. Therefore, transfer from Keepaway to Half-field Offense is complex for traditional approaches, but simple for the BEV. The challenge in representing state in such a way is the consequent high dimensionality. Such high-dimensional structures are effectively trained by indirectly encoding the solution. Indeed, the input and output grid for a fixed BEV substrate is 50×60 to maintain a 1m^2 resolution in the Half-field Offense domain and results in up to 9,000,000 connections, which is computationally prohibitive to directly encode. Through indirect encoding, HyperNEAT-VIRGO can activate only the necessary components, enabling a feasible BEV representation. Thus indirect encoding is pivotal in scaling to complex tasks in large domains.

CHAPTER 8

DISCUSSION AND FUTURE WORK

The HyperNEAT BEV representation exploits indirect encoding and high-dimensional structures to facilitate transfer learning to tasks of increasing complexity. The next section in this chapter discusses the static state representation (e.g. the BEV) and how this representation is beneficial to transfer learning. Then, the chapter later explores the mechanisms that enable such representations and finally contemplates future research directions.

8.1 Representations for Transfer Learning

The BEV shows that a carefully chosen representation with the right encoding can sometimes eliminate the need to change the representation, even across different domains. The BEV facilitates transfer by representing state in a two-dimensional plane, which is suited to tasks such as RoboCup. However, the claim is not that the BEV is a universal solution to task transfer. The ability of a representation to remain static is dependent upon the particular differences between the tasks. Tasks that are semantically similar should be able to represent state information similarly in the BEV, requiring no changes to the representation. Tasks that are significantly different, either through state information or actions, may not allow the representation to remain the same. Methods that alter representation thus remain

important tools in task transfer for domains in which the representation must change with the task. The BEV demonstrates that a carefully chosen representation with the right encoding can sometimes eliminate the need to change the representation, even across different domains. Thus tasks not suited to the BEV representation can still benefit from yet other representations that need not change between them.

The deeper lesson is the critical role of representation in transfer and the consequent need for algorithms that can learn from relatively high-dimensional representations. Indeed, the human eye contains *millions* of photoreceptors, which provide the same set of inputs to every visual task tackled by humans. No new photoreceptor is added for a new task. In effect, visual input to the human eye is a static representation (i.e. it does not alter when changing tasks) of state to the human brain. Yet high-dimensional representations require encodings that take advantage of their expressive power. Indeed, not only does the BEV representation remain static, but the underlying CPPN encoding is not altered between tasks. The mechanisms that enable effective high-dimensional structures are discussed next.

8.2 High-dimensional Structures

This dissertation demonstrated that advanced representations in conjunction with indirect encoding can contribute to scaling learning techniques to more challenging tasks. Advanced representations, such as the BEV, are enabled by *indirect encoding*, which can

effectively train high-dimensional structures. In particular, HyperNEAT exploits geometric *regularities* in the domain to produce optimal weight patterns in ANNs. However, regularities are only one aspect of organization in large structures. This dissertation explored another organizational principle, *modularity*. In the past, HyperNEAT has struggled to produce modularity [CBM10]. The introduction of the Link Expression Output (Chapter 5) allows HyperNEAT to produce modular structures and thereby successfully address tasks in which it has struggled. Such organizational principles from nature may be key to approaching the complexity seen in nature.

Hidden nodes are also an important component in ANNs that address complex domains because an ANN with hidden nodes is able to solve problems that are not linearly separable. Even though the BEV addresses some nonlinearity by creating piece-wise linear approximations, the results from adding hidden layers (Chapter 5) demonstrate that the BEV benefits from the additional nonlinearity provided by hidden nodes. By adding hidden layers, intervening interpretation of the input state, analogous to how the visual cortex interprets data from the eye, is enabled in the BEV representation. A more general insight is that approaches that divide state values into sub-components to solve problems that are not linearly separable may still require nonlinearity to find the optimal solution. Finally, the power of indirect encoding is demonstrated by HyperNEAT’s ability to effectively train high-dimensional structures as quickly as lower dimensional structures (i.e. the BEV with and without hidden layers) because the CPPN is being trained and not the BEV ANN that the CPPN encodes (Chapter 5).

Finally, indirectly encoding the solution enables unique capabilities, such as manipulation of the BEV resolution. Because HyperNEAT indirectly encodes the solution as a function of geometry, the BEV can be trained at one resolution and then extrapolated to different resolutions. This extrapolation enables the unique capability to scale to increased field sizes and improve performance through increased precision without further training. This manipulation also allows the BEV to begin at a smaller resolution that can be incrementally increased over training; however this process sometimes introduces artifacts that hinder the training of the BEV. Ultimately, the indirect encoding expresses geometric patterns at an infinite resolution. Because each object in the BEV is associated with a particular coordinate and the CPPN expresses an infinite resolution function, the Virtual Infinite Resolution by Generation Online (VIRGO) extension to HyperNEAT can dynamically generate nodes at only the precise coordinates required and query an effectively infinite resolution substrate (Chapter 6). Such high-resolution, but computationally manageable, neural substrates potentially begin to approach the complexity of natural neural structures, which can scale machine learning to complex tasks such as RoboCup soccer.

8.3 Implications for Future Work

An exciting implication of this work is that the power of static representations and indirect encoding can potentially bootstrap learning the complete game of soccer. Key ele-

ments of soccer are present in Keepaway and Half-field Offense. Results demonstrate that a static representation can competitively learn to hold the ball in Keepaway and that this skill transfers immediately through the BEV to variations of that task and to Half-field Offense. Interestingly, the Keepaway domain was designed as a stepping stone to scaling machine learning methods to the full RoboCup soccer domain [SS01]. The same principles that enable the BEV to transfer among variations of the Keepaway domain and Half-field Offense *also* can potentially enable the BEV to scale to full Keepaway soccer.

For example, because the representation remains static no matter how many players are on the field, training can begin with a small number of players, such as 3 vs. 3 soccer, and iteratively add more players, eventually scaling up to the full 11 vs. 11 soccer game. Furthermore, varying the substrate configuration while the solution encoding remains static makes it possible to train skills relevant to RoboCup on subsets of the full field, e.g. half-field offense/defense, keepaway, goal shooting, etc. In this way, varying the number of players and varying the field size are *both* required to transfer to full RoboCup soccer. Finally, the static nature of the representation could allow the same policy to train on multiple tasks simultaneously. For example, a soccer player does not practice by playing *only* soccer games. Players improve through multiple drills and continually practice in-between games to refine skills. Thus this dissertation suggests a novel path to learning full-fledged soccer.

A distinctive feature of the BEV representation is that actions are *also* selected in the BEV, that is, the outputs are in the same geometry as the field. In the RoboCup Keepaway domain, actions are constrained to holding the ball and directly passing to a teammate.

However, there are many other actions that are possible, such as clearing the ball, kicking the ball out of bounds, dribbling, and passing to a location close to a teammate. Transfer to Half-field Offense demonstrated that actions could be scaled to include dribbling and shooting on goal. Thus an interesting property of the BEV is that the state space can transfer by accommodating new players or field sizes, and the action space can *also* transfer in the same way. However, such actions are still a small subset of actions available in full soccer. Furthermore, the experiments in this dissertation focused on controlling the player with the ball, but the BEV can potentially control players without the ball. Future approaches to RoboCup soccer can explore controlling all players on the field and expanding the scope of actions from which such players are allowed to select.

A significant aspect of the BEV yet to be explored is extending it to incorporate new or additional information about each object. For example, full RoboCup soccer involves information regarding velocity, stamina, maximum speed, size, power, and many other variables associated with each player. In this dissertation, the BEV only represented the position of each object on the field. Thus transfer that incorporates such additional information may necessarily require additional inputs at each BEV position. However, even then there are potential ways to allow the BEV to retain what was learned in the previous task and build upon that foundation while being altered. For example, one option is to add new input layers or output layers. These new layers can denote new information or actions associated with new objects in the environment while the previously-trained input and output layers retain the prior knowledge. Geometry thus remains an advantage because state information

that is connected with the same location (e.g. all the state data for a single agent) would be located at the same coordinate on separate layers. In contrast, an ANN without such geometry would have no means to discern which original inputs are associated with which new inputs and would thus instead have to learn such relationships. The VIRGO extension to HyperNEAT is also advantageous as the state information for each object is expanded. Instead of creating an increasingly large fixed substrate by increasing the number of input layers and thus increasing the number of superfluous inputs, the VIRGO extension needs only to create the new inputs required at each time step.

Finally, the encoding of the solution also impacts the kinds of policies that are found. For example, in this dissertation the policy is encoded by a CPPN that is expressed as a function of the task geometry, which enables the solution to exploit regularities in the geometry and extrapolate to previously unseen areas of the geometry. It should also be possible to simplify the search for a policy that is a function of the geometry in other learning approaches as well. The challenge is that gradient information (i.e. error) cannot directly pass through the indirection between the ANN and its generating CPPN. A method that solves this problem would open up the power of indirect encoding to all of reinforcement learning.

CHAPTER 9

CONCLUSIONS

This dissertation introduced the bird’s eye view (BEV) representation, which simplifies task transfer by making the state representation static. That way, no matter how many objects are in the domain, the size of the state representation remains the same. In contrast, in traditional representations, changing the number of players (e.g. in the RoboCup Keepaway task) and adding new objects (e.g. a goal) forces changes in the representation by adding dimensions to the state space. In addition to results competitive with leading methods on the Keepaway benchmark, the BEV, which is enabled by an indirect encoding, demonstrated a significant advantage in transfer learning. Transfer proved successful among variations on the number of players and among different field sizes in Keepaway. Also, cross-domain transfer was demonstrated, from Knight Joust to Keepaway. The cross-domain transfer improved not only immediate performance, but also enhanced further learning. Finally, scaling to complex tasks was achieved in transfer from Keepaway to the difficult Half-field Offense domain.

This chapter summarizes the main contributions of the dissertation.

9.1 Contributions

In general, the BEV transfer results highlight the critical role that representation plays in learning and transfer. By altering the representation, transfer learning is simplified and thus validates the hypothesis that representations that need not change between tasks enhance the capability to scale tasks in scope and complexity. The major contributions follow.

1. This dissertation introduced the bird’s eye view (BEV) representation, which is a high-dimensional alternative representation enabled by indirect encoding and trained with HyperNEAT. The BEV exploits the concept of a static representation (i.e. one that need not change) to facilitate transfer between tasks. Because the BEV does not change between tasks, transfer is simplified by recycling the same policy without modification.
2. The BEV was applied to the RoboCup Keepaway benchmark for a comprehensive study that compares it to traditional approaches in machine learning and transfer learning. In directly learning the popular 3 vs. 2 Keepaway benchmark, the HyperNEAT-trained BEV achieves competitive performance against traditional approaches such as Sarsa, NEAT, and EANT. In transfer learning, the BEV provides a significant performance advantage over the popular TVITM-PS approach to transferring between tasks. This transfer advantage applies in variations of the Keepaway task (i.e. increasing the number of players) and cross-domain transfer (i.e. Knight Joust to Keepaway).

3. HyperNEAT was extended with the Link Expression Output (LEO) to enable the discovery of modular ANN weight patterns by optimizing connectivity separately from weight values. The power of this extension was demonstrated in the Retina Problem domain, a task for which traditional HyperNEAT failed in producing solutions because of an inability to create modular weight patterns. Thus HyperNEAT-LEO encapsulates two significant natural organizational principles: regularity and modularity. These principles are essential to producing effective high-dimensional structures, such as those required by the BEV.
4. The BEV’s ability to address nonlinearity was evaluated through a comparison of the BEV with and without hidden layers. The BEV without hidden layers addresses some nonlinearity through a piecewise linear approximation, thus allowing the BEV to approximate a nonlinear function. However, the investigation into nonlinearity demonstrated that the BEV benefits by the nonlinearity provided by hidden nodes. Thus such approximations may require further nonlinearity to solve problems optimally.
5. The unique capability to scale the size of the BEV through indirect encoding was explored. First, scaling to larger field sizes and increasing performance instantaneously by increasing the size of the BEV was shown in the RoboCup Keepaway benchmark, without further training. Next, the Virtual Infinite Resolution by Generation Online (VIRGO) extension to the HyperNEAT algorithm was introduced. In this extension, HyperNEAT dynamically generates only the required input and output nodes at each time step and then queries the CPPN for connection weights between them. Thus

HyperNEAT-VIRGO generates effectively infinite-resolution ANN substrates, thereby improving performance in the RoboCup Keepaway benchmark. Finally, the technique of *phased continuous substrate extrapolation*, whereby the resolution of the ANN substrate is set to low resolution at the beginning of training and gradually increased in resolution during training, was investigated. This approach demonstrated promise in decreasing the computational cost, but ultimately created distortions in the training process that negatively impacted performance.

6. Finally, HyperNEAT-trained Keepaway policies were transferred to the RoboCup Half-field Offense domain. This experiment demonstrated that the BEV representation is a vehicle for transfer to increasingly complex tasks and potentially provides an avenue for bootstrapping into full RoboCup soccer.

Thus this dissertation highlights that representation plays a critical role in machine learning and in transfer learning, where it is not only important to learn a task, but also to apply learned knowledge to a new task. In particular, representations that need not change during transfer may be the ideal approach to transfer learning, such as the high-dimensional BEV representation. The role of representation in transfer is relevant to all approaches to learning because transfer is always an option for extending the scope of learning. Thus encoding research, such as in generative and developmental systems [HP02, Lin68, Sta07, Tur52], and representation research, such as in relational reinforcement learning [DRD01, Mor03, TGD04], is important to machine learning in general.

9.2 Conclusion

This dissertation introduced the BEV representation, simplifying task transfer through a state representation that need not change between tasks. In this way, the BEV facilitates transfer among the Knight Joust, RoboCup Keepaway, and RoboCup Half-field Offense domains. Furthermore, such high-dimensional static representations require indirect encodings that take advantage of their expressive power, such as in HyperNEAT. The hope is that advanced representations in conjunction with indirect encoding can later contribute to scaling learning techniques to more challenging tasks, such as the complete RoboCup soccer domain.

APPENDIX A:

C# ROBOCUP SIMULATOR

The RoboCup domain is a powerful and popular research tool, but there are also significant barriers to machine learning research in the domain. The RoboCup Soccer simulator server alleviates the barriers by removing the burden of physical robots and providing a platform and language-agnostic means of research [NNM98]. The server and players communicate through a server/client architecture, where the soccer simulator (server) and players (clients) are independent from each other. Because the simulator communicates over a network interface, players may be implemented in any platform that supports such network communication. Thus the server and players can run independently on different architectures and different software. This platform-independent implementation allows varied learning systems to be compared in the same domain without reimplementing. Players receive updates from the server identically, regardless of implementation, for sensory information. Players also manipulate actions in the server through a common set of commands. This standardization creates a common platform for research. Furthermore, the server maintains the essential challenges of RoboCup, such as agent architecture design, sensor fusion, strategic planning, and multi-agent coordination.

Building upon the RoboCup server’s success, the dissertation work included designing a new implementation of the RoboCup simulator suited for batch experimentation. This new simulator closely emulates the original RoboCup Soccer server [NNM98]. However instead of distributing agents across the network, they run within the simulator, thereby removing the need to transmit and parse messages. This design provides a significant speed-up (up to ten times) over the original RoboCup simulator. Additionally, the adapted simulator can be run

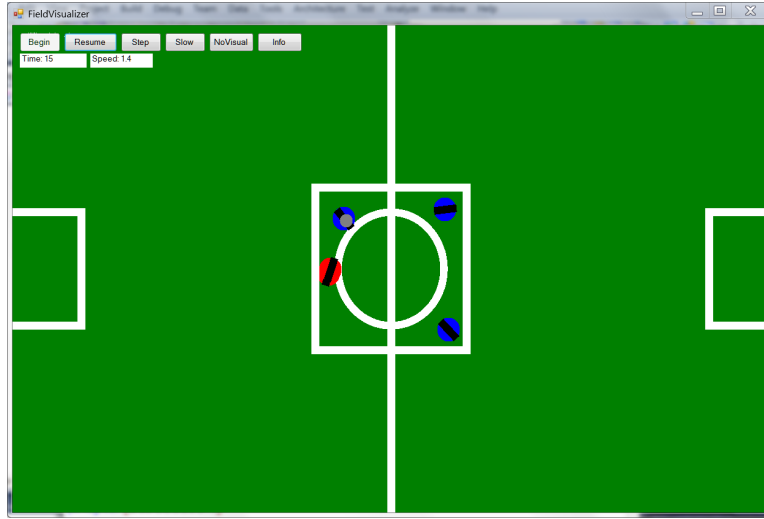


Figure A.1: Visualization of the RoboCup Field in the Newly-Implemented Simulator. The visualizer for the new RoboCup simulator is built-in and may be optionally run at program startup. The visualization draws the field, the players (red and blue colors indicate side), and the ball as a gray circle. The visualization can begin, pause, step the simulation, and disable on-screen animation to reduce processing overhead. The simulator displays current simulator time, that is, the number of cycles executed, and the current simulation speed, which is the multiple of simulated cycles executed in one-tenth of a second. The new simulator provides a fast alternative platform for experimenting with learning in RoboCup.

within an experimental framework, instead of in a separate process, allowing the simulator to be controlled as needed by the experiment, such as pausing, resetting, or manipulating objects. Because the new simulator conforms to the .NET Framework Common Language Specification (CLS), it guarantees cross-platform and cross-language compatibility. This guarantee means that any other .NET programming language can import the new simulator into methods implemented in that language. Finally, the combined server implementation includes a built-in visualizer for the field (figure A.1) and information about specific objects. This new simulator is a practical contribution to the RoboCup research community.

APPENDIX B: PARAMETER VALUES

This appendix contains the parameter settings for the experiments described in this dissertation. Because HyperNEAT differs from original NEAT only in its set of activation functions, it uses the same parameters [SM02]. Experiments were run using an implementation of HyperNEAT in C# written by the author (available at <http://eplex.cs.ucf.edu>). Parameters were found to be robust to moderate variation through preliminary experimentation. Similar parameters are used in all experiments, with a few minor exceptions. This section also provides a detailed explanation of each parameter.

1. *Population Size*: The number of networks evaluated every generation.
2. c_1 : A parameter from the original NEAT that defines the weight of excess genes when computing compatibility between networks.
3. c_2 : A parameter from the original NEAT that defines the weight of disjoint genes when computing compatibility between networks.
4. c_3 : A parameter from the original NEAT that defines the weight of connection strength differences when computing compatibility between networks.
5. c_t : A parameter from the original NEAT that defines the compatibility threshold that determines whether networks are in the same species. In the implementation in this dissertation this value is variable and changes to accommodate the desired number of species.
6. $p(\textit{Add Link})$: The probability of adding a new connection to a network (i.e. to the CPPN).

7. *p(Add Node)*: The probability of adding a new node to a network (i.e. to the CPPN).
8. *# Species*: The desired number of species in the population. If there are more or less species, c_t will be adjusted down or up respectively in an effort to maintain the target number of species.
9. *Elitism*: The top percentage of each species that will be copied into the next generation unchanged.
10. *Threshold*: The value that the CPPN's output must exceed for a connection to be expressed.
11. *Weight Range*: The range of weights that the CPPN can assign to the connections in the substrate. The output of the CPPN will be scaled between these values.
12. *Functions*: The set of functions from which the CPPN can choose from when adding a node.

The parameters c_1 , c_2 , and c_3 are set to 1.0 for all experiments and the value c_t starts at 20.0, but varies dynamically during evolution. Elitism is set to 10% for all experiments. Each experiment's weight range is $[-3.0, 3.0]$ and function sets consisting of Gaussian, bipolar sigmoid, sine, absolute value, step, and linear. Values for parameters that vary by experiment can be seen in Table A.1. The population varies depending on the computational complexity of the domain and the target number of species varies with the population to maintain an average of 10-15 individuals per species. Mutation rates also vary with population size.

Table A.1: Parameter Settings in Experiments

<i>Parameter</i>	<i>Keepaway Benchmark</i>	<i>Modularity Joust</i>	<i>Modularity Retina</i>	<i>Modularity Keepaway</i>
Pop. Size	100	150	500	100
p(Add Link)	0.18	0.18	0.1	0.15
p(Add Node)	0.05	0.05	0.03	0.05
# Species	10	15	30	10
Threshold	0.2	variable	variable	variable
<i>Parameter</i>	<i>Nonlinearity</i>	<i>Resolution Field Size</i>	<i>Resolution VIRGO</i>	<i>Resolution Phased</i>
Pop. Size	100	100	100	100
p(Add Link)	0.15	0.18	0.15	0.15
p(Add Node)	0.05	0.05	0.05	0.05
# Species	10	10	10	10
Threshold	LEO	0.2	LEO	LEO
<i>Parameter</i>	<i>Half-field</i>			
Pop. Size	100			
p(Add Link)	0.15			
p(Add Node)	0.05			
# Species	12			
Threshold	LEO			

Consequently smaller populations receive higher mutation rates to ensure a reasonable number of structural changes are made each generation. Finally, threshold values vary between experiments performed pre-LEO and post-LEO.

LIST OF REFERENCES

- [Aal09] Aaltonen et al. (over 100 authors). “Measurement of the Top Quark Mass with Dilepton Events Selected using Neuroevolution at CDF.” *Physical Review Letters*, **102**(15):2001, 2009.
- [Alt94] Lee Altenberg. “Evolving Better Representations through Selective Genome Growth.” In *Proceedings of the IEEE World Congress on Computational Intelligence*, pp. 182–187, Piscataway, NJ, 1994. IEEE Press.
- [ASP00] Peter J. Angeline, Gregory M. Saunders, and Jordan B. Pollack. “An Evolutionary Algorithm That Constructs Recurrent Neural Networks.” 2000. In press.
- [BK99] Petet J. Bentley and S. Kumar. “Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, pp. 35–43, 1999.
- [Bon02] Josh C. Bongard. “Evolving Modular Genetic Regulatory Networks.” In *Proceedings of the 2002 Congress on Evolutionary Computation*, 2002.
- [BW93] Heinrich Braun and Joachim Weisbrod. “Evolving Feedforward Neural Networks.” 1993.
- [Car97] Rich Caruana. “Multitask learning.” In *Machine Learning*, pp. 41–75, 1997.
- [CBM10] J. Clune, B.E. Beckmann, P.K. McKinley, and C. Ofria. “Investigating whether HyperNEAT produces modular neural networks.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2010)*, New York, NY, 2010. ACM Press.
- [CBO09] Jeff Clune, Benjamin E. Beckmann, Charles Ofria, and Robert T. Pennock. “Evolving Coordinated Quadruped Gaits with the HyperNEAT Generative Encoding.” In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2009) Special Session on Evolutionary Robotics*, Piscataway, NJ, USA, 2009. IEEE Press.
- [CDF03] Mao Cheny, Klaus Dorer, Ehsan Foroughi, Fredrik Heintz, ZhanXiang Huangy, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Jan Murray, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, Yi Wangy, and Xiang Yin. *Robocup Soccer Server: User’s Manual*. The Robocup Federation, 4.00 edition, February 2003.
- [Cla89] P. Clark. *Machine and Human Learning*. London: Kogan Page, 1989.

- [CLL09] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. “On-line Neuroevolution Applied to The Open Racing Car Simulator.” In *Proceedings of the 2009 IEEE Congress on Evolutionary Computation (IEEE CEC 2009)*, Piscataway, NJ, USA, 2009. IEEE Press.
- [CSP11] Jeff Clune, Kenneth O. Stanley, Robert T. Pennock, and Charles Ofria. “On the Performance of Indirect Encoding Across the Continuum of Regularity.” *IEEE Transactions on Evolutionary Computation*, 2011.
- [CW08] Ronan Collobert and Jason Weston. “A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning.” In *Proceedings of the 25th International Conference on Machine Learning*, New York, NY, 2008. ACM Press.
- [DM92] Dipankar Dasgupta and Douglas McGregor. “Designing Application-Specific Neural Networks Using the Structured Genetic Algorithm.” In *Proceedings of the International Conference on Combinations of Genetic Algorithms and Neural Networks*, pp. 87–96, 1992.
- [DRD01] Saïçeo Diçeroski, Luc De Raedt, and Kurt Driessens. “Relational Reinforcement Learning.” *Machine Learning*, **43**(1-2):7–52, April-May 2001.
- [DS07] David D’Ambrosio and Kenneth O. Stanley. “A Novel Generative Encoding for Exploiting Neural Network Sensor and Output Geometry.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007)*, New York, NY, 2007. ACM Press.
- [DS08] David B. D’Ambrosio and Kenneth O. Stanley. “Generative Encoding for Multiagent Learning.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008)*, New York, NY, 2008. ACM Press.
- [DS10] David D’Ambrosio and Kenneth O. Stanley. “Evolving Policy Geometry for Scalable Multiagent Learning.” In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2010)*, p. 8, New York, NY, USA, 2010. ACM Press.
- [Egg97] P. Eggenberger. “Evolving Morphologies of Simulated 3D Organisms Based on Differential Gene Expression.” pp. 205–213, 1997.
- [FDM08] Dario Floreano, Peter D’ÁErr, and Claudio Mattiussi. “Neuroevolution: from architectures to learning.” *Evolutionary Intelligence*, **1**(1):47–62, 2008.
- [FFP90] D. B. Fogel, L. J. Fogel, and V. W. Porto. “Evolving neural networks.” *Biological Cybernetics*, **63**(6):487–493, 1990.

- [GGW04] Ivan Garibay, Ozlem Garibay, and Annie Wu. “Effects of module encapsulation in repetitively modular genotypes on the search space.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, pp. 1125–1137, New York, NY, USA, July 2004. ACM.
- [GM99] Faustino Gomez and Risto Miikkulainen. “Solving Non-Markovian Control Tasks with Neuroevolution.” pp. 1356–1361, 1999.
- [GS07] Jason Gauci and Kenneth O. Stanley. “Generating Large-Scale Neural Networks Through Discovering Geometric Regularities.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007)*, New York, NY, 2007. ACM Press.
- [GS08] Jason Gauci and Kenneth O. Stanley. “A Case Study on the Critical Role of Geometric Regularity in Machine Learning.” In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*, Menlo Park, CA, 2008. AAAI Press.
- [GS10a] Jason Gauci and Kenneth O. Stanley. “Autonomous Evolution of Topographic Regularities in Artificial Neural Networks.” *Neural Computation*, **22**(7):1860–1898, 2010.
- [GS10b] Jason Gauci and Kenneth O. Stanley. “Indirect Encoding of Neural Networks for Scalable Go.” In *Proceedings of the 11th International Conference on Parallel Problem Solving From Nature (PPSN 2010)*, Berlin, 2010. Springer.
- [GWP96] Frederic Gruau, Darrell Whitley, and Larry Pyeatt. “A Comparison Between Cellular Encoding and Direct Encoding for Genetic Neural Networks.” In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 81–89. MIT Press, 1996.
- [Har93] Inman Harvey. *The Artificial Evolution of Adaptive Behavior*. PhD thesis, School of Cognitive and Computing Sciences, University of Sussex, Sussex, 1993.
- [HHL99] L.H. Hartwell, J.H. Hopfield, S Leibler, and A.W. Murray. “From molecular to modular cell biology.” *Nature*, **402**, 1999.
- [HP02] Gregory S. Hornby and Jordan B. Pollack. “Creating High-Level Components with a Generative Representation for Body-Brain Evolution.” *Artificial Life*, **8**(3), 2002.
- [IJC89] David J. Montana and Lawrence Davis. “Training Feedforward Neural Networks Using Genetic Algorithms.” pp. 762–767, 1989.

- [IJC91] Alexis Wieland. “Evolving Neural Network Controllers for Unstable Systems.” pp. 667–673, 1991.
- [KA05] Nadav Kashtan and Uri Alon. “Spontaneous evolution of modularity and network motifs.” *Proceedings of the National Academy of Sciences*, **102**(39):13773–13778, September 27 2005.
- [KAK97] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. “RoboCup: A Challenge Problem for AI.” *AI Magazine*, **18**(1):73–87, Spring 1997.
- [KGB05] Vadym Kyrylov, Martin Greber, and David Bergman. “Multi-criteria optimization of ball passing in simulated soccer.” *Journal of Multi-Criteria Decision Analysis*, **13**:103–113, 2005.
- [KLS07a] Shivaram Kalyanakrishnan, Yaxin Liu, and Peter Stone. “Half Field Offense in RoboCup Soccer: A Multiagent Reinforcement Learning Case Study.” In Gerhard Lakemeyer, Elizabeth Sklar, Domenico Sorenti, and Tomoichi Takahashi, editors, *RoboCup-2006: Robot Soccer World Cup X*, volume 4434 of *Lecture Notes in Artificial Intelligence*, pp. 72–85. Springer Verlag, Berlin, 2007.
- [KLS07b] Shivaram Kalyanakrishnan, Yaxin Liu, and Peter Stone. *RoboCup 2006: Robot Soccer World Cup X*, volume 4434 of *Lecture Notes in Computer Science*, chapter Half Field Offense in RoboCup Soccer: A Multiagent Reinforcement Learning Case Study. Springer Berlin / Heidelberg, 2007.
- [KM11] Nate Kohl and Risto Miikkulainen. “An Integrated Neuroevolutionary Approach to Reactive Control and High-level Strategy.” *IEEE Transactions on Evolutionary Computation*, 2011.
- [KSJ00] Eric R. Kandel, James H. Schwartz, and Thomas M. Jessell. *Principles of Neural Science*. McGraw-Hill, New York, fourth edition, 2000.
- [KSV05] Jelle R. Kok, Matthijs T. J. Spaan, and Nikos Vlassis. “Non-communicative multi-robot coordination in dynamic environments.” *Robotics and Autonomous Systems*, **50**(2-3):99–114, February 2005.
- [Kui00] Benjamin Kuipers. “The Spatial Semantic Heirarchy.” *Artificial Intelligence*, **119**:191–233, 2000.
- [Lin68] A. Lindenmayer. “Mathematical Models for Cellular Interaction in Development Parts I and II.” *Journal of Theoretical Biology*, **18**:280–299 and 300–315, 1968.
- [Lip07] H. Lipson. “Principles of modularity, regularity, and hierarchy for scalable systems.” *Journal of Biological Physics and Chemistry*, **7**, 2007.

- [Mac09] Alan Mackworth. “Agents, Bodies, Constraints, Dynamics, and Evolution.” *AI Magazine*, **30**(1):7–28, Spring 2009.
- [Mar99] Andrew P. Martin. “Increasing Genomic Complexity by Gene Duplication and the Origin of Vertebrates.” *The American Naturalist*, **154**(2):111–128, 1999.
- [MEK07] Jan FH. Metzen, Mark Edgington, Yohannes Kassahun, and Frank Kirchner. “Performance Evaluation of EANT in the RoboCup Keepaway Benchmark.” In *ICMLA ’07: Proceedings of the Sixth International Conference on Machine Learning and Applications*, pp. 342–347, Washington, DC, USA, 2007. IEEE Computer Society.
- [MM96] David E. Moriarty and Risto Miikkulainen. “Efficient Reinforcement Learning Through Symbiotic Evolution.” *Machine Learning*, **22**:11–32, 1996.
- [MM97] David E. Moriarty and Risto Miikkulainen. “Forming Neural Networks Through Efficient and Adaptive Co-Evolution.” *Evolutionary Computation*, **5**:373–399, 1997.
- [Mor03] Eduardo Morales. “Scaling up reinforcement learning with a relational representation.” In *Proceedings of the Workshop on Adaptability in Multi-agent Systems (AORC-2003)*, pp. 15–26, Sydney, Australia, January 2003.
- [MP88] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, expanded edition, 1988.
- [NNM98] Itsuki Noda, C Fl Itsuki Noda, Hitoshi Matsubara, Hitoshi Matsubara, Kazuo Hiraki, Kazuo Hiraki, Ian Frank, and Ian Frank. “Soccer server: A tool for research on multiagent systems.” *Applied Artificial Intelligence*, **12**:233–250, 1998.
- [PP98] Joao Carlos Figueira Pujol and Riccardo Poli. “Evolving the Topology and the Weights of Neural Networks Using a Dual Representation.” *Applied Intelligence Journal*, **8**(1):73–84, January 1998. Special Issue on Evolutionary Learning.
- [PY08] Sinno Pan and Qiang Yang. “A Survey on Transfer Learning.” Technical Report HKUST-CS08-08, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, November 2008.
- [Rad93] Nicholas J. Radcliffe. “Genetic Set Recombination and its Application to Neural Network Topology Optimization.” *Neural computing and applications*, **1**(1):67–90, 1993.
- [RDC07] Jan Ramon, Kurt Driessens, and Tom Croonenborghs. “Transfer Learning in Reinforcement Learning Problems Through Partial Policy Recycling.” In *Proceedings of the 18th European Conference on Machine Learning*, pp. 699–707, Berlin, Germany, 2007. Springer-Verlag.

- [RN94] G. A. Rummery and M. Niranjan. “On-line Q-learning using connectionist systems.” CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994.
- [SBM05] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. “Real-Time Neuroevolution in the NERO Video Game.” *IEEE Transactions on Evolutionary Computation Special Issue on Evolutionary Computation and Games*, **9**(6):653–668, 2005.
- [SDG09] Kenneth O. Stanley, David B. D’Ambrosio, and Jason Gauci. “A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks.” *Artificial Life*, **15**(2):185–212, 2009.
- [SF95] N. Saravanan and David B. Fogel. “Evolving Neural Control Systems.” *IEEE Expert*, pp. 23–27, June 1995.
- [SI94] Jurgen Schmidhuber and Fakultat Fur Informatik. “On learning how to learn learning strategies.” Technical report, Fakultat fur Informatik, Technische Universitat Munchen. Revised, 1994.
- [SKT06] Peter Stone, Gregory Kuhlmann, Matthew E. Taylor, and Yaxin Liu. “Keepaway soccer: From machine learning testbed to benchmark.” In *RoboCup-2005: Robot Soccer World Cup IX*, pp. 93–105. Springer Verlag, 2006.
- [SM02] Kenneth O. Stanley and Risto Miikkulainen. “Evolving Neural Networks Through Augmenting Topologies.” *Evolutionary Computation*, **10**:99–127, 2002.
- [SM03] Kenneth O. Stanley and Risto Miikkulainen. “A Taxonomy for Artificial Embryogeny.” *Artificial Life*, **9**(2):93–130, 2003.
- [SM04] Kenneth O. Stanley and Risto Miikkulainen. “Competitive Coevolution Through Evolutionary Complexification.” **21**:63–100, 2004.
- [SMS06] Frieder Stolzenburg, Jan Murray, and Karsten Sturm. “Multiagent matching algorithms with and without coach.” *Journal of Decision Systems*, **15**(2-3):215–240, 2006. Special issue on Decision Support Systems. Guest editors: Fatima C. C. Dargam and Pascale Zarate.
- [SS01] Peter Stone and Richard S. Sutton. “Scaling Reinforcement Learning Toward RoboCup Soccer.” In *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.
- [SSK05] Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. “Reinforcement Learning for RoboCup-Soccer Keepaway.” *Adaptive Behavior*, **13**(3):165–188, 2005.

- [SSS01] Peter Stone, Richard S. Sutton, and Satinder Singh. “Reinforcement Learning in 3 vs. 2 Keepaway.” In Peter Stone, T. Balch, and G. Kraetszchmar, editors, *Robocup-2000: Robot soccer world cup IV*, pp. 249–258. Springer Verlag, Berlin, 2001.
- [Sta07] Kenneth O. Stanley. “Compositional Pattern Producing Networks: A Novel Abstraction of Development.” *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems*, **8**(2):131–162, 2007.
- [Sut96] Richard S. Sutton. “Generalization in reinforcement learning: Successful examples using sparse coarse coding.” In *Advances in Neural Information Processing Systems 8*, pp. 1038–1044. MIT Press, 1996.
- [Tad08] Prasad Tadepalli. “Learning to Solve Problems from Exercises.” *Computational Intelligence*, **4**(24):257–291, 2008.
- [TGD04] Prasad Tadepalli, Robert Givan, and Kurt Driessens. “Relational reinforcement learning: An overview.” In *International Conference on Machine Learning Workshop on Relational Reinforcement Learning*, New York, NY, 2004. ACM Press.
- [Thi96] Dirk Thierens. “Non-Redundant Genetic Coding of Neural Networks.” In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 571–575, 1996.
- [TM94] Sebastian Thrun and Tom M. Mitchell. “Learning one more thing.” Technical report, Carnegie Mellon University, 1994.
- [TS07a] Erik Talvitie and Satinder Singh. “An Experts Algorithm for Transfer Learning.” In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pp. 1065–1070, 2007.
- [TS07b] Matthew E. Taylor and Peter Stone. “Cross-Domain Transfer for Reinforcement Learning.” In *Proceedings of the 24th international conference on Machine learning*, pp. 879–886, New York, NY, USA, 2007. ACM.
- [TS09] Matthew E. Taylor and Peter Stone. “Transfer Learning for Reinforcement Learning Domains: A Survey.” *Journal of Machine Learning Research*, **10**, 2009.
- [TSL07] Matthew E. Taylor, Peter Stone, and Yaxin Liu. “Transfer Learning vis Inter-Task Mappings for Temporal Difference Learning.” *Journal of Machine Learning Research*, **1**(8):2125–2167, September 2007.
- [TSW08] Lisa Torrey, Jude W. Shavlik, Trevor Walker, and Richard Maclin. “Rule Extraction for Transfer Learning.” In *Rule Extraction from Support Vector Machines*, pp. 67–82. Springer-Verlag, Berlin, Germany, 2008.

- [TTM08] Lisa Torrey, Trevor Walker, Richard Maclin, and Jude Shavlik. “Advice Taking and Transfer Learning: Naturally Inspired Extensions to Reinforcement Learning.” In *AAAI Fall Symposium on Naturally Inspired AI*, Washington, DC, 2008. AAAI Press.
- [Tur52] A. M. Turing. “The Chemical Basis of Morphogenesis.” *Royal Society of London Philosophical Transactions Series B*, **237**:37–72, August 1952.
- [TWS06] Matthew E. Taylor, Shimon Whiteson, and Peter Stone. “Comparing Evolutionary and Temporal Difference Methods in a Reinforcement Learning Domain.” In *GECCO 2006: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1321–1328, July 2006.
- [TWS07] Matthew E. Taylor, Shimone Whiteson, and Peter Stone. “Transfer via Inter-Task Mappings in Policy Search Reinforcement Learning.” In *The Autonomous Agents and Multi-Agent Systems Conference*, New York, NY, May 2007. AAMAS-2007, ACM Press.
- [VS10a] Phillip Verbancsics and Kenneth O. Stanley. “Evolving Static Representations for Task Transfer.” *Journal of Machine Learning Research (JMLR)*, **11**:1737–1769, 2010.
- [VS10b] Phillip Verbancsics and Kenneth O. Stanley. “Task Transfer through Indirect Encoding.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)*, New York, NY, 2010. ACM Press.
- [VS11] Phillip Verbancsics and Kenneth O. Stanley. “Constraining Connectivity to Encourage Modularity in HyperNEAT.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011)*, New York, NY, 2011. ACM Press.
- [WA96] G.P. Wagner and L. Altenberg. “Complex adaptations and the evolution of evolvability.” *Evolution*, **50**, 1996.
- [WAG09] R. Paul Wiegand, Gautham Anil, Ivan Garibay, Ozlem Garibay, and Annie Wu. “On the Performance Effects of Unbiased Module Encapsulation.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2009)*, pp. 1729–1736, New York, NY, USA, July 2009. ACM.
- [WDD93] Darrell Whitley, Stephen Dominic, Rajarshi Das, and Charles W. Anderson. “Genetic Reinforcement Learning for Neurocontrol Problems.” *Machine Learning*, **13**:259–284, 1993.
- [Whi95] D. Whitley. “Genetic Algorithms and Neural Networks.” In J. Periaux, M. Galan, and P. Cuesta, editors, *Genetic Algorithms in Engineering and Computer Science*, pp. 203–216. 1995.

- [Whi05] Shimon Whiteson. “Improving reinforcement learning function approximators via neuroevolution.” In *AAMAS ’05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pp. 1386–1386, New York, NY, USA, 2005. ACM.
- [WHR87] James D. Watson, Nancy H. Hopkins, Jeffrey W. Roberts, Joan A. Steitz, and Alan M. Weiner. *Molecular Biology of the Gene Fourth Edition*. The Benjamin Cummings Publishing Company, Inc., Menlo Park, CA, 1987.
- [WKM05] Shimon Whiteson, Nate Kohl, Risto Miikkulainen, and Peter Stone. “Evolving Soccer Keepaway Players Through Task Decomposition.” *Mach. Learn.*, **59**(1-2):5–30, 2005.
- [WSB90] Darrell Whitley, T. Starkweather, and C. Bogart. “Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity.” *Parallel Computing*, **14**:347–361, 1990.
- [WTS09] Shimon Whiteson, Matthew E. Taylor, and Peter Stone. “Critical Factors in the Empirical Performance of Temporal Difference and Evolutionary Methods for Reinforcement Learning.” *Journal of Autonomous Agents and Multi-Agent Systems*, **21**(1):1–27, 2009.
- [WW07] Shimon Whiteson and Daniel Whiteson. “Stochastic Optimization for Collision Selection in High Energy Physics.” In *IAAI 2007: Proceedings of the Nineteenth Annual Innovative Applications of Artificial Intelligence Conference*, July 2007.
- [Yao99] Xin Yao. “Evolving Artificial Neural Networks.” *Proceedings of the IEEE*, **87**(9):1423–1447, 1999.
- [ZBL99] Michael J. Zigmond, Floyd E. Bloom, Story C. Landis, James L. Roberts, and Larry R. Squire, editors. *Fundamental Neuroscience*. Academic Press, London, 1999.